# Improving modularity of interactive software with the MDPC architecture

Stéphane Conversy[1,2], Eric Barboni[2], David Navarre[2] & Philippe Palanque [2]

[1] ENAC – Ecole Nationale de l'Aviation Civile
7, avenue Edouard Belin, 31055 Toulouse, France.
stephane.conversy@enac.fr

[2] LIIHS – IRIT, Université Paul Sabatier
118 route de Narbonne, 31062 Toulouse Cedex 4, France
{barboni, conversy, navarre, palanque}@irit.fr
http://liihs.irit.fr/{barboni, navarre, palanque}

**Abstract.** The "Model - Display view - Picking view - Controller" model is a refinement of the MVC architecture. It introduces the "Picking View" component, which offloads the need from the controller to analytically compute the picked element. We describe how using the MPDC architecture leads to benefits in terms of modularity and descriptive ability when implementing interactive components. We report on the use of the MDPC architecture in a real application: we effectively measured gains in controller code, which is simpler and more focused.

**Keywords:** MVC, interactive software, modularity, Model Driven Architecture

## 1 Introduction

Modularity is an aspect of software engineering that helps improve quality and safety of software: once designed, implemented, and verified, modular components can be reused in multiple software so that such software can rely on their soundness. The advent of rich interaction on the web, and the advent of WIMP interaction in airplane cockpits [1][2] raise interest in interactive software architecture. The need to use, develop, and extend toolkits for interaction makes programmers eager to study this area. Similarly, a number of widgets have been formally described, so as to comply with important properties of interactive systems [14]. As a toolkit programmer point of view, reusing these components would ensure that his particular implementation complies with the same properties.

*Separation of concerns* is a design principle that can help to achieve modularity: the idea is to break a problem into separate sub-problems and design software components that would handle each sub-problem. The Model-View-Controller (MVC) architecture is a well-known attempt to improve modularity of software [18] through separation of concerns (cf Fig. 1). In MVC, the Model encapsulates the data to be interacted with, the View implements the graphical representation and is

updated when the Model changes, and the Controller translates actions from the user to operations on the Model. MVC has been successfully applied to high-level interactive components, though in this form it resembles more to the PAC architecture than its original description [6]. For example, frameworks to help develop interactive application, such as Microsoft MFC, organize the data structure in a document, and views on the document that are updated when the document changes. When applied to very low-level interactive components though, such as scrollbars, programmers encounter difficulties to clearly modularize the components so that the original goal of reusing components is reached: the View and the Controller components of the widget are so tightly coupled that it seems useless and a waste of time to separate them, as they cannot be reused for other interactive widgets[1].

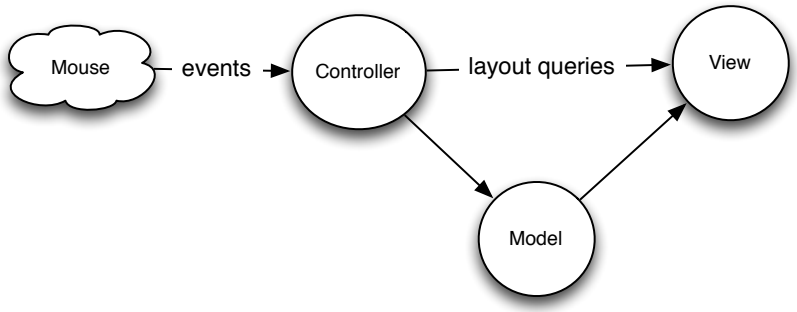



**Fig. 1:** MVC: The controller queries the view to know which part of the view has been clicked in order to react accordingly.

We argue in this paper that by externalizing the *picking concern* from the Controller, we can actually modularize a set of interactive widgets so that the Controller can be reused across different classes of Views of the same model. We first present the causes of the problem mentioned above. We then introduce the Model – Display view – Picking view – Controller (MDPC) architecture, and show with examples how to use it. We then report our experience at refactoring a real application with the MDPC model.

## 2 The need to externalize Picking

At its lowest level, today's interactions usually involve a rasterized image (i.e. a digital/sampled/pixel-based image) and a pointer that the user controls to point at a given pixel. *Rendering* is the process of transforming a logical description or the *conceptual* model of an interactive component to a graphical representation or a *perceptual* model. *Picking* can be considered as the inverse process of rendering:

[1] As stated by the designers of JAVA Swing: "We quickly discovered that this split didn't work well in practical terms because the view and controller parts of a component required a tight coupling (for example, it was very difficult to write a generic controller that didn't know specifics about the view). So we collapsed these two entities into a single UI (user-interface) object […]". http://java.sun.com/products/jfc/tsc/articles/architecture/#roots

*Picking* is the process of determining/querying the graphical primitives that colored/filled a given pixel, and in turn the corresponding conceptual entity. Usually, interactive systems use the pixel underlying the cursor, in order to react when the user clicks on an interactive component. Picking is also used during passive movements, for example to determine when the cursor enters an interactive component so as to highlight it.

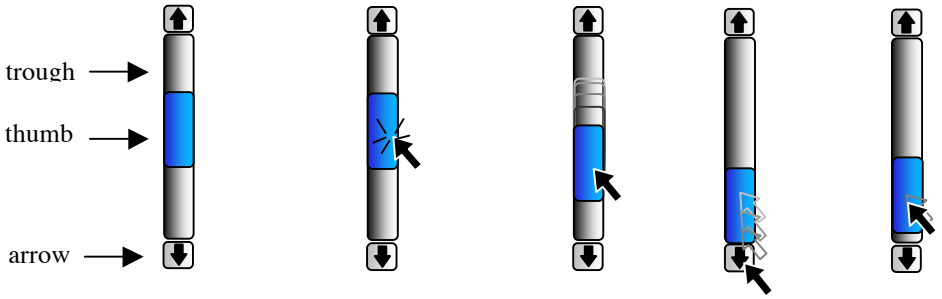



**Fig. 2:** a scrollbar and its parts

For the remaining of this section, we take the scrollbar as an example (Fig. 2). A scrollbar is an instrument that enables a user to specify the location of a range by direct manipulation. For example, the user can use a scrollbar to modify the position of the view of a document too large to be displayed at once. Conceptually, a scrollbar is composed of four parts: a *thumb* to control the position of a range of values, a *trough* in which the user can drag the thumb, i.e. the position of the thumb is constrained inside the trough, and two *arrows* for decrementing/incrementing the position of the thumb by a fixed amount.

```
if( (event.y > y_up_widget) and (event.y <
  y_bottom_widget) { // test if it is in the widget
  if (event.y < y_up_widget+harrow) {
    // scroll down by one line
    ...
  } else if (event.y<ythumb) {
    // scroll down by one viewing area
} else //...and so on
```

**Fig. 3**: An example of code using analytic picking

In the original form of MVC, the Controller usually handles picking by receiving low-level events such as mouse clicks or mouse moves. For example, if the user clicks in the image of a scrollbar for a text editor document, the Controller computes which part of the view has been clicked on, and calls a particular method of the Model with a computed parameter: if the part is one of the arrows, the Controller sets the Model's value by decreasing or increasing it by an amount equivalent to that of one line. If the part is the space between the thumb and the arrows, the amount is equivalent to that of one viewing area. In order to determine the part that has been clicked on, the Controller must know the layout of the widget parts, *i.e* the location of parts that are displayed on the screen [15]. For example, with a vertical scrollbar, if the upper ordinate of the widget is $y_{widget}$, the height of an arrow is $h_{arrow}$, and the upper

ordinate of the thumb is $y_{thumb,}$ a Controller can determine which part has been clicked on by using the code in Fig. 3.

The code is embedded into the method that reacts to the click on the view. This prevents modularization of the controller: it is specially designed for one particular view, even if some of the values can be parameterized, such as the location of the whole widget. In particular, the relative layout of the different parts of the widget is often hard-coded, and is not a parameter of the widget.

In fact, most interactive widgets are structured around parts that embody a spatial mode of interaction i.e. a same event in two different parts lead to two different behaviors of the widget. For example, clicking in an arrow triggers a different action than the one corresponding to clicking in the thumb. In a part, the action triggered by an event is the same regardless of the parameters of the event. Only the parameters of the action may depend on the dimensions of the event. What is important then to implement part-dependant code, is not the low level parameters of events such as the x and y coordinates, but the part on which the event took place. Thus, the Controller behavior must be dependant on parts below the cursor, and not the cursor's x and y position, so that the code that describes it would resemble to code in Fig. 4.

```
if( isin(event, scrollbar)) { // test if it is in the widget
   if (isin(event,uparrow)) {
      // scroll down by one line
      ...
   } else if (isin(event,thumb)) {
      // scroll down by one viewing area
      ...
   } else { //...and so on
      ...
   }
}
```

**Fig. 4**: An example of controller code independent of the exact position of parts

In this case, the "*isin*" function is a call to an external picking function. As such it is a mean to factor out the picking process from the Controller, and enables its reuse with other Views. However, implementing the controller with multiple *if/then/else* prevents extension and combination, as adding a part requires adding code to handle it. Instead, we propose to completely externalize the picking process, and make the Controller behavior dependant on *Leave*/*Enter* events, instead of *Move* events.

Usually, programmers describe graphics by the mean of graphical shapes: instead of filling pixels by themselves, they use a higher level of description, for example a circle at a given position with a given radius. A graphical library in turn fills the pixels according to the description. A *Leave* event is triggered when the shape under the cursor changes between two consecutive *Move* events. A *Leave* event is immediately followed by an *Enter* event, as leaving a shape means that the cursor enters another shape (we consider the background as a shape with infinite size, which lies under every other shape). *Leave* and *Enter* events are synthesized events: they are computed from *Mov*e events, and a description of the layout and contours of the shapes in used. Thus, *Leave*/*Enter* events generation requires a data structure that keeps track of the layout of the shapes and their contours. This kind of data structure

is called a scene-graph. Usually, a scene-graph is used as an intermediate stage in the rendering process described above: the programmer describes the rendering of the conceptual model in terms of shapes, their geometrical and styling transformation, that are stored in a scene-graph. Since a scene-graph knows about the layout and contours of shapes, it is able to determine the shape that is under the cursor. Thus a scene-graph can handle input and implement a picking service, as well as synthesize *Leave*/*Enter* events.

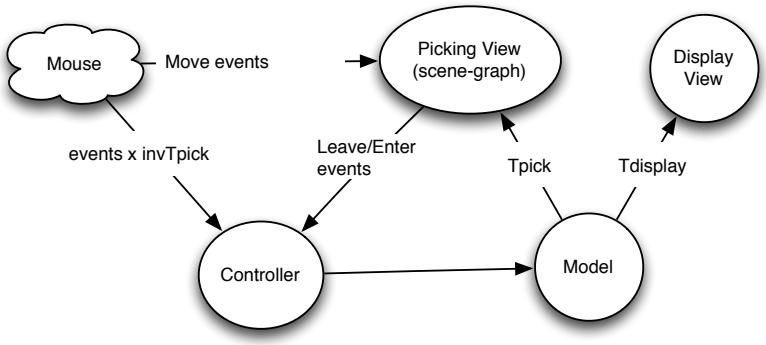



**Fig. 5:** The Model – Picking View – Display view Controller (MDPC) architecture

**Display View and Picking View**

We propose to split the View component into two components: the Display View, which is exactly the ancient MVC View, and the Picking View, which is an invisible rendering of the model that is specialized to facilitate interaction description. By splitting them, we deepen the separation of concerns aspect of the MVC model: while the display view handles the representation that has to be perceived by the user, the Picking View helps the determination of the part of the widget that is under the cursor. This separation also solves two problems of the design of interactive widgets, related to the differences between the structure of the graphics for interaction and the structure of the graphics for display: those due to graphic design concerns, and those due to transient, invisible interactive structure.

When developing widgets, a programmer can use graphical primitives that do not fit with interaction needs. For example a scrollbar can be seen as a thumb moving into a trough (Fig. 2). This can be described as two shapes, the thumb shape lying on top of the trough shape. If this structure were mapped to a scene graph, the Enter and Leave event would contain the identifier of the shapes, regardless of the position of the cursor relative to the thumb. Thus, the Controller would receive the same Move event, be it above the thumb, or below the thumb, and would not be able to discriminate the zone in which the cursor has actually entered (above or below the thumb), though this information is mandatory to implement control. This fact usually leads the programmer to implement analytic code, i.e. code that uses the x and y position of the thumb to eventually determine the zone. However, if the design of the view is done with three parts, the "decrease part", the thumb, and the "increase part", the only

information required to implement the interaction is the part identifier dimension of the Enter/Leave events. It is therefore necessary to decouple the display part of a widget from its picking part. Furthermore, interactive projects now involve graphic designers, whose creativity may be refrained by coding requirements. The separation between display and picking frees the graphic designer from the obligation to respect a graphical structure that does not map with the desired display: would the display view serves for both display and picking, the designer is required to use a three parts graphic, although two parts would have been enough. On the other hand, a designer can use as many graphical primitives she needs (like soft shadows, filters etc.), and in any configuration. In particular, she could have used sub-shapes like text of other images useful for the user to understand the display, but that are of no interest for interaction. As unnecessary graphical structures increases the complexity of formal checking of the controller code, reducing their number is important.

We saw above that the picking structure can be different from the display structure. But it can also change for the sake of interaction, while the display structure remains the same. In the scrollbar example, when one clicks on the thumb to move it along the trough, there are invisible zones in which spatial mode of interaction enters in action (Fig. 2, right). When the thumb "hits" the top or the bottom of the trough, the thumb does not move even if the user goes on with his movement, as the thumb is constrained in the borders of the widgets. However, when the user reverses his movement, there is a position from which the thumb starts moving again. This position is invisible, but can be computed as soon as the user clicks in the thumb: in the case of the vertical scrollbar, it is equal to the position of the widget plus the difference between the click and the top of the thumb. When the cursor is in this zone, the thumb moves as the cursor moves. When the cursor leaves this zone, and enters one of the two other zones, the thumb position is not updated anymore (and is set to 0 or 1). Usually, the interaction is described by using a "special mode" of the controller: as soon as the user clicks on the thumb, the controller "captures" the cursor so that moving it on top of unrelated display areas will not trigger associated actions. This behavior is traditionally implemented with analytic determination of distance from important points, such as the one described above. Instead, we propose to implement it using the same mechanism outlined above, namely with zones that are entered and left, with the difference that this time they are invisible and transient, as they are enabled only in certain states of the widgets. Hence, for one model, there can be one displayed view, whatever the interaction handled by the widget, and two different invisible, transient views to implement control, which leads to the split between Display Views and Picking Views.

## 3 Example 1: the scrollbar in depth

In this section we show how to use the MDPC model to describe the scrollbar. The model of the scrollbar is a range whose two boundaries lie in the range from 0 to 1 (Fig. 6). To specify values in an arbitrary range of values, not only 0 to 1, we can use a linear (i.e. $ax+b$) transform function when notifying observers. The Scrollbar widget enables a user to specify position of the range, i.e. she can slide the range so that both

boundaries are changed at the same time. The extent of the range (i.e. the difference between the boundaries) is specified by either the application, or is tied to another widget such as a text widget. The range-slider is a scrollbar widget, augmented with instruments that enable the user to specify the values of the boundaries. Hence, the Scrollbar and the Range-slider share the same model.

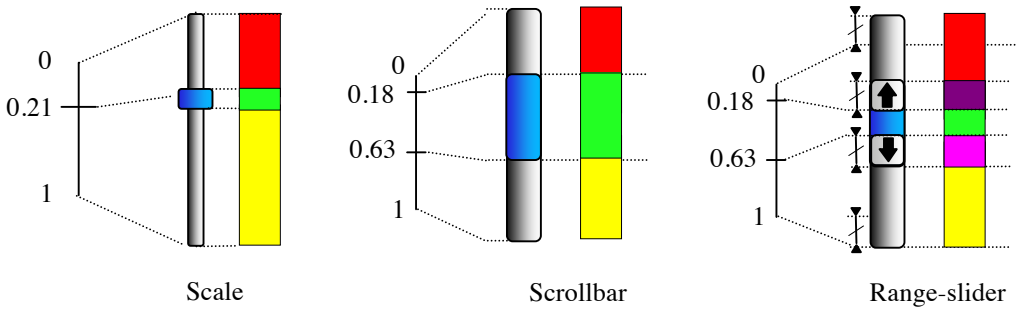



**Fig. 6**: From left to right, the Model, the Display View, and the Picking View of the Scale, the Scrollbar, and the Range-slider. The model of the Scrollbar and the Range-slider is the same.

The display view is a drawing composed of several graphical shapes. In its simplest usable form, the drawing may resemble to Fig. 2: one background shape for the trough, and one shape for the thumb, lying over the background shape. The size of the trough is arbitrary chosen. The size of the thumb can be computed from the values of the model and the size of the trough, using a simple linear function. However, the thumb has a minimum size to allow the user to pick it regardless of the extent it is supposed to reflect. As explained above, the structure of the display view cannot be used to implement the control, as it is necessary to differentiate between the part of the trough that is above the thumb from the part that is below. Hence, the picking view is composed of three shapes, one for the thumb, and two for the visible parts of the trough. When the user manipulates the thumb, the position of the thumb shape is changed in the display view and in the picking view, while the size of the two sub-shapes of the trough are changed in the picking view. The controller of the scrollbar can then be described with events that contain the identifier of the shapes, as there is no need to analytically compute which part has been clicked on.

**Invariance to geometrical transform and relative layout transform**

The horizontal scrollbar is a 90° rotated vertical scrollbar. As such, it can be implemented by adding a 90° rotation in the display view and the picking view components. The interaction corresponding to a click in the arrows, and in the two parts of the trough, is exactly the same. However, in traditional MVC, the controller code of the vertical scrollbar has to be updated to handle the new positions of the parts. The controller as we defined it, does not need to be changed for a vertical scrollbar: it is invariant with respect to geometric transforms. This result is true for one type of interaction with the scrollbar, namely clicks in part that triggers action.

With the 90° rotation example, the vertical movement corresponding to the manipulation of the trough is not compatible with the orientation of the scrollbar. To overcome this problem, we use the inverse transform that enables the generation of the view, by transforming the events so that their coordinates are relative to the view, and not absolute (or relative to the screen). Using the inverse transforms, the controller remains the same.

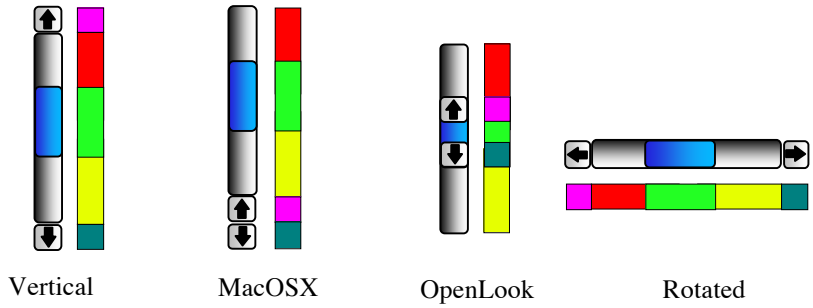



**Fig. 7**: The Display View, and the Picking View of varieties of Scrollbar.

Moreover, the controller is invariant with respect to the relative layout of parts of the scrollbar. As shown on SEQ, the arrows can be move at one extremity of the trough (to mimic a variety of MacOSX scrollbar), or even at the ends of the thumb (to mimic OpenLook scrollbar). The same MDPC controller as the vertical scrollbar can control these kinds of scrollbar, whereas with MVC each variant requires a different controller.

### Multiple picking views for transient behavior

When sliding the trough though, the user can go outside the widget and still hold control of the scrollbar. This has been handled in traditional architecture with a special mode of interaction, namely by "capturing" the cursor so that any other underlying system such as MVC is bypassed. With our model, moving outside the widget will trigger a Leave event, and eventually stops the controller. This behavior is due to the fact that dragging the thumb is actually a completely different interaction than clicking in scrollbar parts. In fact, the picking model is different from the one described above. The sliding interaction is dependent on three zones: one in which moving the cursor moves the thumb (or more precisely, set the boundaries of the scrollbar model, which is reflected by the view as a displacement of the thumb), and two in which movement has no effect on the model (and hence on the view of the thumb) because the thumb hit one of the edges of the trough. This can be implemented as another picking view (see SEQ, left). When clicking on the thumb, the "waiting-for-click" picking view of SEQ is replaced by the "sliding" picking view. When the cursor moves inside the central part, the thumb follows its position. When the cursor enters the upper part, the value of the Model is set to 0, and does not move until it reenters the central area again. As long as the user holds the button

pressed, the controller receives Leave, Enter and Move events and reacts accordingly. When the user releases the cursor, the "waiting-for-click" picking view comes back.

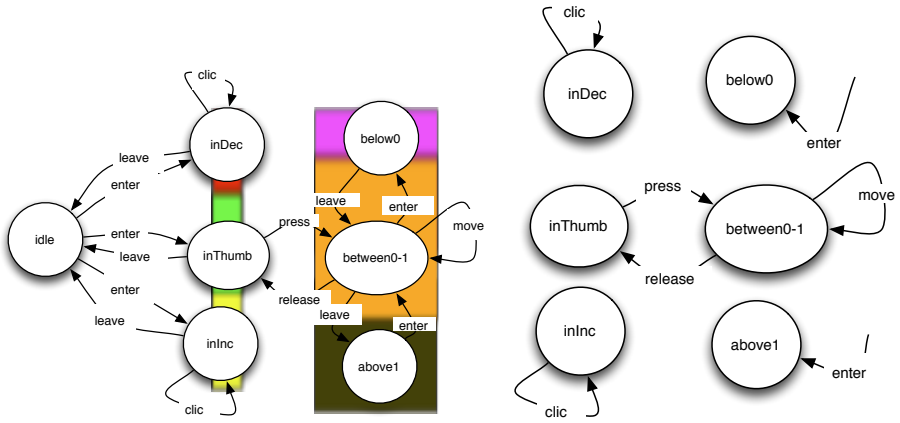



**Fig. 8:** To the left, the state-machine describing the behavior of the scrollbar. To the right, a simplified version with the transitions with associated actions only.

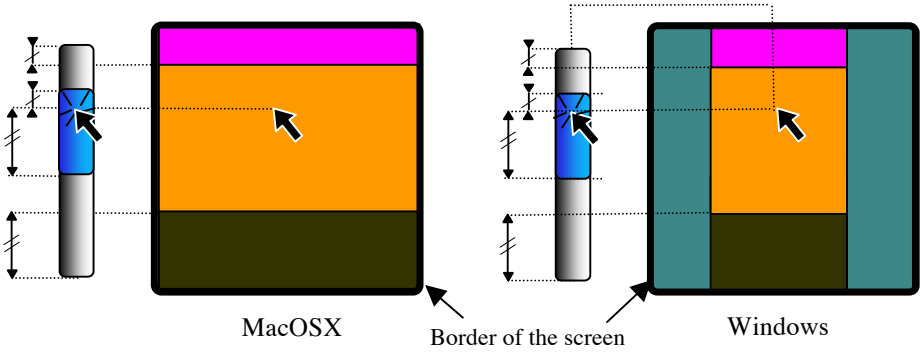



**Fig. 9:** when clicking on the thumb, a new Picking View is used. The thick rounded rectangle reflects the border of the screen.

To assess the universality of this model, we can describe a variation of this interaction. While sliding the thumb, the user can move the cursor at a particular distance from the scrollbar. With a MacOSX scrollbar, this distance is infinite, and can be described with rectangular zones that extent horizontally up to the border of the screen. With a Windows scrollbar, the distance is finite, and when crossed, the thumb goes back to the position it has at the beginning of the interaction (i.e. when the user clicks on the thumb), enabling the user to cancel the interaction. This can be described by shrinking the three zones of the picking view (SEQ, right), so that the background appears at their sides: when the cursor enters the background zone, the Controller resets the thumb position back to its previous value.

## 4 Example 2: the bar chart and the pie chart

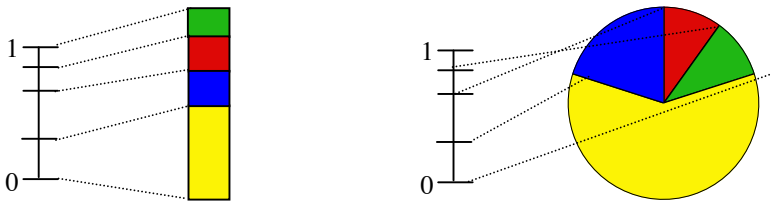



**Fig. 10**: the Model of the partition and two Display Views: a Pie Chart, and a Bar Chart.

A partition model can be considered as a list of floats that range from 0 to 1. Each pair of floats specifies an interval. It can be represented with a bar chart, in which each part's height is proportional to its interval. It can also be represented with a pie chart, in which the extent angle of each part is proportional to its interval. Charts are often used as visualization only. However, a user can specify the values by clicking and dragging the borders between each part. Fig. 11 shows a picking view that enables this interaction. Thick borders reflect the interactive parts. They might be invisible in the display view, but are necessary to ease interaction. When clicking on a border, the second picking view enters in action, and precludes the user to move a value below or above neighbor values. It seems difficult to use the same controller for both Bar and Pie picking views since they are so different. However, they are topologically equivalent. We can use the inverse of the transform that generates the view: the Bar view involves a transformation from Cartesian coordinates, while the Chart view involves a transformation from polar coordinates.

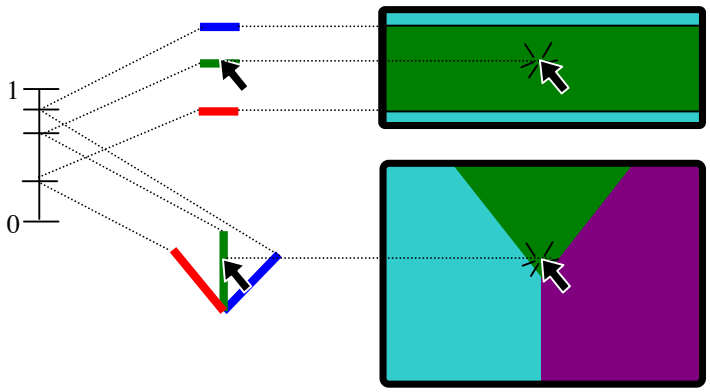



**Fig. 11:** Above: the "wait-for-click" Picking View and "sliding" Picking View of the Bar Chart. Below: the "wait-for-click" Picking View and sliding Picking View of a Pie Chart. Both "sliding" picking view prevent the user to move a value below or above neighbor values.

## 5 Example 3: the hierarchical menu

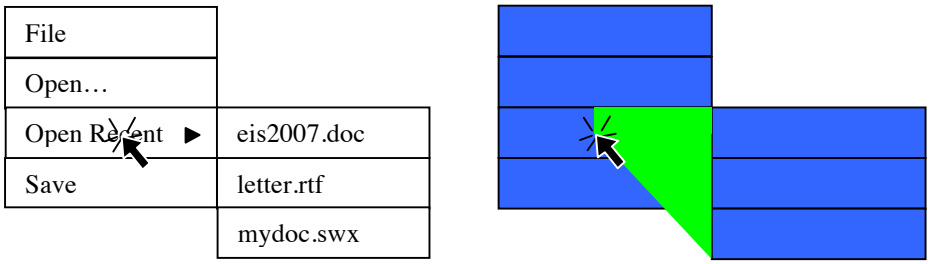



**Fig. 12:** the Display View and Picking View of a deployed hierarchical menu.

When clicking on an entry of a hierarchical menu that has sub-entries, a pull-down menu shows up. On MacOSX, the controller allows the user to "fly over" entries of the first menu and reach entries of the submenu that are displayed at the bottom and left of the current location of the cursor. If the cursor goes downward vertically, it enters another entry, and the sub-menu hides itself. If the user does not initiate the interaction after a few milliseconds, the "fly over" mode is stopped. As shown in Fig. 12, it can be implemented with a transient triangular-like shape in the Picking View. Apart from the fact that the MDPC model eases the comprehension of the behavior, it leads again to less code in the controller, as no analytical computation is necessary to implement control. Moreover, it simplifies the architecture of the code, since no special mode of interaction in which the cursor is captured is necessary. It also shows that the picking structure can be very different from the display structure: it needs a transient state in which a shape helps implement interaction, but that is hidden to the developer. Finally, the set of necessary shapes for picking are less important than the set necessary for display (for each entry in the hierarchical menu: a sub-shape for the background, the text, the triangle icon). When using the same scene-graph for both display and picking, special code that prevents action for Leave/Enter events involving sub-shapes is needed. Separating the scene-graphs removes this obligation, and leads to smaller, more focused, code.

## 6 Return of experience with a real application

We updated the architecture of a real application that uses the ARINC 661 set of widgets [2]. The purpose of ARINC 661 specification [1] is to define interfaces to a Cockpit Display System used in interactive cockpits. MPIA is an airborne application that uses the ARINC 661 specification, and that aims at handling several flight parameters. It is made up of 3 pages (called WXR, GCAS and AIRCOND) between which crewmember are allowed to navigate using 3 buttons (as shown on Fig. 13). Interaction with MPIA relies on button-like widgets exclusively. Though we did not use the button as an example in previous sections, our observation that controller code is polluted by picking code holds true: with the previous architecture, picking is done by traversing the tree of widgets and by checking for each widget whether it is picked

We want to show with this example that externalizing the picking process leads to more simpler, more focused code.

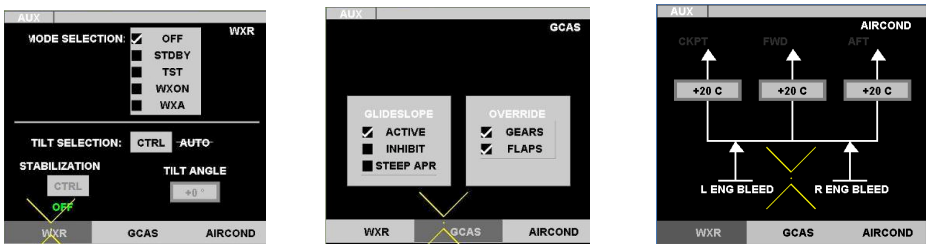



**Fig. 13**: the three pages of MPIA

Though we described control with state machines so far, for this application we used the ICO formalism [17], which is based on Petri-Nets. The next section describes how rendering is done using declarative specifications, and how the renderer implements picking services for the handling of low-level user input events.

## Rendering

In the previous version, rendering was implemented with Java code, using the Java2D API. Instead, we now rely on an SVG description. SVG is an xml-based vector graphics format: it describes graphical primitives in terms of analytical shapes and transformations. As such, SVG is a scene-graph. To render SVG, we use the Batik renderer. Transforms from models to graphics are done with XSLT. XSLT is an xml-based format that describes how to transform an xml description (the source) to another xml description (the target). An XSLT description is called a "stylesheet". XLST is traditionnaly used in batch mode to transform a set of xml files, but XSLT can also be used in memory so that performances are compatible with interaction. We used the Apache Xalan library to handle XSLT transforms.

In our case, the source is a DOM description of the components the application: the "ARINC tree". It is built at startup time, together with the instantiation of the ICOs components. Before running the application, the system compiles two stylesheets to two XSLT transformers: one for the display view, and one for the picking view (Fig. 14). This compilation can be triggered at any time, to update a stylesheet while designing and implementing it. While running the application, each time the state of an ARINC tree variable changes, the transformer transforms the ARINC tree to two DOM SVG trees, which in turn are passed to the SVG renderer (Fig. 16). The display view is then displayed in a window, while the picking view is rendered in an offscreen window.

Each time the cursor moves on the display view, the picking manager "picks" the topmost graphical item *of the picking view* at the position of the cursor, as if the cursor was moving over the picking view instead of the display view. Then, the picking manager sends an event to the Petri nets with the cursor position and the ID of

the graphical item under the cursor as parameters. The Petri nets specification then uses the ID to retrieve the instance of the models over which the cursor is.

```
arinc xml description:
<arinc>
  <button x="10" y="10" width="200" height="50" text=”submit”
enable=”1”/>
</arinc>

xslt stylesheet:
<xsl:stylesheet>
          <xsl:template name="button">
                    <rect x="{@x}" y="{-(@y+@height)}" width="{@width}"
height="{@height}" rx="150" fill="url(#gradientPanelBackground)”/>
                    <text x="{@x}"  y=”{-@y}">submit</text>
</xsl:stylesheet>

generated svg:
<rect x="100" y="-60" width="200" height="50" rx="150"
fill="url(#gradientPanelBackground)”/>
<text x="100" y="-50">submit</text>
```

Fig. 14. Examples of an ARINC tree, an XSLT transformer, and the resulting SVG Picking

### Advantages of the architecture

Our goal with this application is to show that it is possible to externalize picking from the controller. The resulting refactoring first shows that the architecture is implementable, and that it enabled us to reduce the complexity of the controller code by a significant amount (about 25% less), without removing functionality. While applying it to the entire modeling of the MPIA application and the user interface server compliant with ARINC 661 specification this produced a significant reduction of model size as shown in Fig. 15**.** This difference is more salient with widgets in charge of the assembly of widgets as the ones shown Fig. 15**.** For other terminal widgets (like command buttons, text boxes), the reduction of models size is still present but more limited.

| Widget | Model size without MDPC | | Model size with MDPC | |
|---|---|---|---|---|
| | Places | Transitions | Places | Transitions |
| RadioBox | 49 | 29 | 28 | 21 |
| TabbedPanelGroup | 62 | 22 | 44 | 16 |
| TabbedPanel | 72 | 49 | 22 | 7 |
| Panel | 65 | 46 | 16 | 5 |

**Fig. 15.** Measure of volume of each widget in terms of model size

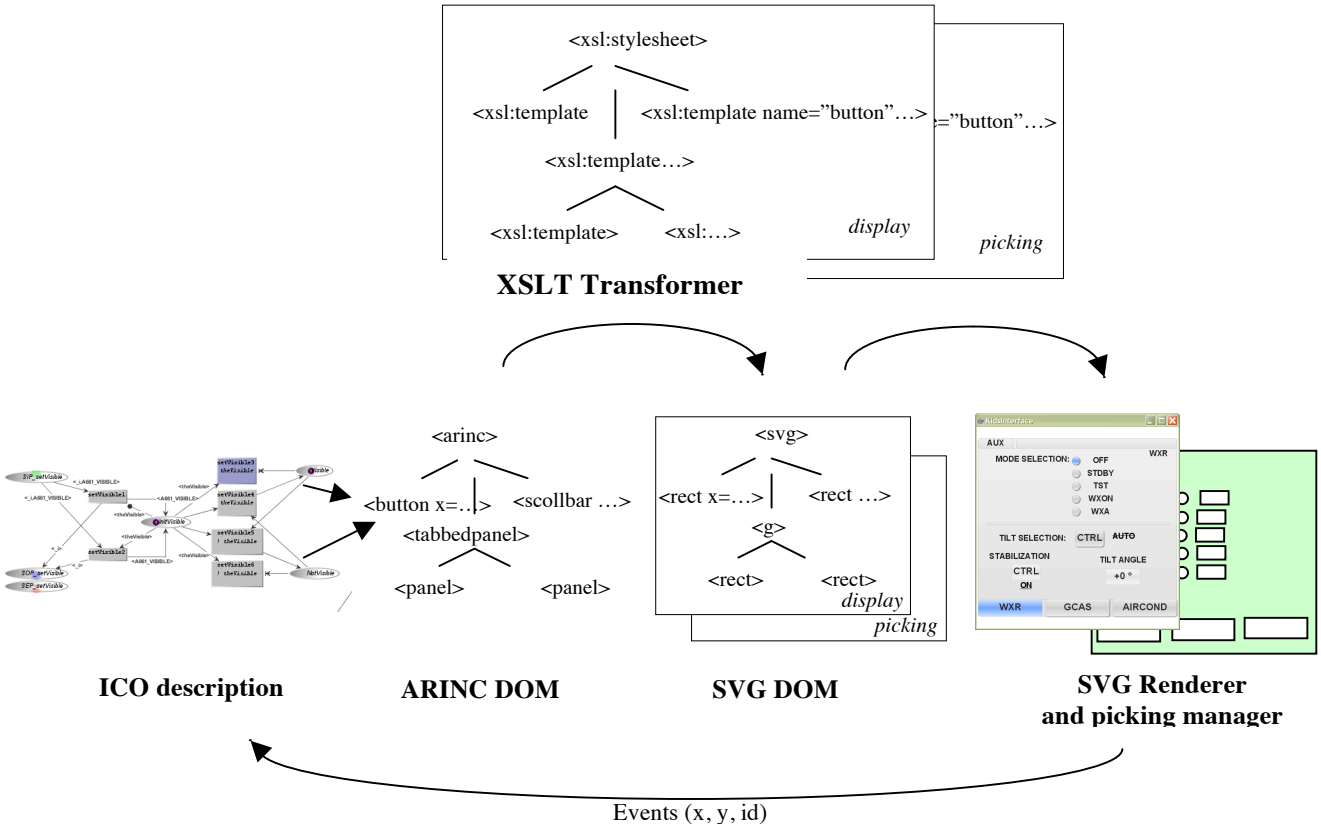



**Fig. 16:** The run-time architecture

This architecture clearly distinguishes the conceptual model from the perceptual model, and gathers all the graphics and transforms description into one external entity. It has clear advantages over the previous architecture. First, it increases readability of the graphical code. Second, changing the look of an application is as simple as changing the XSLT file. Fig. 17 shows two renderings that can be alternatively presented without making any change in the models describing the widgets. Most commercial drawing and painting software can produce SVG graphics compatible with our system, allowing graphic artists to be involved earlier in the design process of interactive applications [5]. Finally, rendering is considered as a transformation process that uses functional programming without side-effect, which increases robustness [9]. It is also interesting to note that the advantages of the architecture are demonstrated both at the level of programming code and at the level of model description.

**Drawbacks of the new architecture**

The performance of dedicated Java2D code is much better than the one exploiting SVG, XSLT, and Batik. The low performance of the new architecture comes from the fact that the transformation process involves the entire conceptual model each time it is triggered, leading to a whole new SVG DOM, even if a single variable of the ARINC DOM has changed. This problem is related to the current status of transforming tools, which are unable to do incremental transformations. An incremental transformer is able to only update the changing parts of the target tree, which increases performances (up to 500 times) [16][22]. Another solution is to use "active transformations", i.e. transformation systems and specifications designed to implement incremental transformations [2].


**Fig. 17**: The same User Application window with two different stylesheets

## 7 Related work

Fabrik is a direct manipulation - based user interface builder that enables a designer to specify transforms between widget with a visual flow language [10]. Events flow in the same flow graph that describes the geometrical transforms, so that they are automatically transformed to a position relative to the graphically transformed widget.

In [7], Dragicevic and Fekete introduce the MVzmC architecture. Like MDPC, the widget is divided into zones that embody a spatial mode of interaction. The "view controller" plays the role of our transform mechanism, and works similarly to Fabrik transforms. However, the Vzm component is still in charge of determining which zone has been hit, hence it is not invariant to changes of relative layout of parts. Similarly, in the Event-driven MVC [20], the code that handles picking is buried into the view, and hence precludes any simple change of layout. MDPC clearly factors out this task from the Controller and the View, which leads to more reusable code. Finally, both the MVzmC and Event-driven MVC use a single view, and cannot be used to implement transient picking structure.

Using a declarative description of an interface is not new (see [21] for a review). However, in much of these systems, the description is only a way to get the interface outside the code of the application: a run-time environment displays widgets by interpreting the description. Furthermore, the description is only about the layout of

predefined WIMP widgets at the finer level of details. Such systems are primarily targeted at toolkit users (i.e. interactive application designers) that do not need to implement new or slightly different interaction techniques. In our case, the architecture describes all models, from the level of the application down to the inner mechanics of a widget. For example, we can describe the control and the rendering of a range slider using the same architecture that we use to describe the application, while it's not possible with other systems.

The idea of transforming a conceptual model to a perceptual model comes from the Indigo project [4], a novel client-server architecture for highly interactive systems. While X11 splits rendering and interaction between the server and the client, Indigo makes the server in charge of the rendering *and* the interaction. To reflect changes of the logic of the client into the rendering, Indigo uses a transformation process similar to the one described in this paper. Indigo widgets are part of the server, and the rendering is not done using a transformation process. In our architecture, we apply the transformation model to the inner mechanics of the widgets. Transforms are also used in [13], but it is done once at the instantiation of widgets from a layout description, while transforms are used continuously in our architecture.

## 8 Discussion

This work attempts to tackle the question often asked when disserting about the MVC model: what is a controller exactly? As inventors of MVC apply separation of concerns to interaction code, we can apply separation of concerns down to the Controller itself: in MVC, the controller handles *picking*, the backward *translation* of dimension of events to arguments for operations on the model, and the *management* of the interactive state of interactive components (as opposed to graphical state). In MDPC, the combination of the scene-graph (the picking view) and Leave/Enter events synthesis handles picking. The picking code is hence offloaded from the Controller code, which makes the controller simpler. In order to pass computed values from events to arguments for operations on the model, the old Controller has to transform dimensions of the events in the widget referential: hence, it is dependent on the View, as it must queried its parameters (such as the orientation) to compute the inverse transform. This backward translation from the dimensions of the events to arguments for operation on the model can be handled by the inverse transform mechanism in the MDPC model. We have shown how to do it functionally with rotated scrollbar and pie charts. If this translation is more complex, it can be dealt with with a similar mechanism to MVzmC's one, i.e. a View Controller. Hence, in the MDPC architecture, what we call a controller is the piece of code that manages the interactive state of a component, i.e. the state-machine or the Petri Net that describes it. The interactive state is different from the graphical state. The graphic state is just a direct translation from the model to a graphical representation. For example, if a scrollbar is disabled because the interface does not allow the user to interact with it, there should be a Boolean in the model that should reflect it, and that would be used to draw a disabled toolbar (for example in gray). The management of interactive state is the core functionality of the Controller: it defines the behavior, or the inter-actions

between the user and the model, i.e. the intertwined sequences of actions from the user, and actions from the system that change the set of future actions at user's disposal. With such a definition, Controllers presented in this paper seem simple. However, when dealing with multiple inputs, the description is complex, and may require Petri Nets with dozens of places and transitions. With the MPIA application, the code associated to transition is limited to change of values in the model, without any other computations. Hence the Controller is the Petri net, and almost nothing else, except the rules that change values in the model. In other words, it seems to us that it is impossible to remove other aspects of the Controller, as we reduced it to its crux.

Another goal of this project was to foster the use of an MDA approach to the design of interactive application. We designed the models of our widgets in order to make them as independent as possible from controllers and views, which led to the merge of the scrollbar model and the range slider model into a single range model. The choice of setting the bounds of the values inside the models to a range of [0,1] makes the model even more reusable, since the addition of a linear function makes it general enough to describe previous use of scrollbars. Our approach is an attempt to maximize the late binding aspect of our components: MDPC makes use of late binding of range bounds, of positions, and of relative position of parts.

## 9 Conclusion

In this paper, we have presented a new architecture for interactive systems implementation. We split the View component of MVC in a Picking View and Display View components. The picking task, traditionally handled by controllers of interactive widgets, is offloaded to a picking manager, which turns Move events to Leave/Enter events by using the picking view. Widgets following this architecture gain invariance from relative layout of components and invariance from geometrical transforms. The Controller code shrinks and is more focused to its functional core. The architecture can also be used to implement invisible, transient interactive structure. One of the goal of this project is to have a complete MDA driven widget set. The MDPC architecture is a first step towards this objective, as it enables the definition of interactive systems based on a MDA approach. The controller is specified using a formalism such as Petri Nets, the display and picking view are specified with a transformation model based on declarative specifications. In order to fully accomplish our goal, we need better and more efficient transform tools. In particular, we plan to design incremental, and bidirectional transform engine, in order to ease the definition of transforms. Another result is more conceptual: thinking of control as leaving/entering/moving over/clicking on possibly invisible parts helps design and describe it, as shown in the hierarchical menu example.

**Acknowledgments.** This work is partly funded by DPAC (Direction des Programmes de l'Aviation Civile) étude "validation cockpit interactif" and by EU via the Network of Excellence ResIST (www.resist-noe.org).

## References


1. ARINC 661 specification: Cockpit Display System Interfaces To User Systems, Prepared by Airlines Electronic Engineering Commitee, Published by Aeronautical Radio, 2002.
2. Barboni, E., Conversy, S., Navarre, D. & Palanque, P. Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification. Proc of DSVIS 2006, Lecture Notes in Computer Science, Springer Verlag
3. Beaudoux O., 2005. XML Active Transformation (eXAcT): Transforming Documents within Interactive Systems. Proc of the 2005 ACM Symposium on Document Engineering (DocEng 2005), ACM Press, pages 146-148.
4. Blanch R., Beaudouin-Lafon, M., Conversy, S., Jestin, Y., Baudel, T. and Zhao, Y. P. INDIGO : une architecture pour la conception d'applications graphiques interactives distribuées. In Proceedings of IHM 2005, pages 139-146, Toulouse - France, Sept. 2005
5. Chatty, S., Sire, S., Vinot, J., Lecoanet, P., Lemort, A., and Mertz, C. 2004. Revisiting visual interface programming: creating GUI tools for designers and programmers. In Proceedings of UIST '04. ACM Press, New York, NY, 267-276
6. Coutaz, J. PAC, an Object Oriented Model for Dialog Design, in Proc. of Interact'87 (North Holland, 1987), 431-436.
7. Dragicevic P. and Fekete J-D. Étude d'une boîte à outils multi-dispositifs. Proc. of the 11th French speaking conf. on Human-Computer Interaction (IHM'99), pages 33-36, 1999
8. Extensible Markup Language (XML) 1.0 (Third Edition) W3C Recommendation http://www.w3.org/TR/REC-xml/
9. Hudak, P. 1989. Conception, evolution, and application of functional programming languages. ACM Comput. Surv. 21, 3 (Sep. 1989), 359-411.
10. Ingalls, D., Wallace, S., Chow, Y., Ludolph, F., and Doyle, K. 1988. Fabrik: a visual programming environment. In Proc of OOPSLA (San Diego, California, United States, September 25 - 30, 1988). ACM Press, New York, NY, 176-190.
11. Jacob, R. J. 1996. A Visual Language for Non-WIMP User Interfaces. In Proc. of Symposium on Visual Languages (1996). VL. IEEE Computer Society, Washington, 231.
12. Krasner, G. E. and Pope, S. T. 1988. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. J. Object Oriented Program. 1, 3 (Aug. 1988), 26-49.
13. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Víctor López Jaquero, UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004), Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.
14. Navarre David; Palanque Philippe; Bastide Rémi, and Sy Ousmane. Structuring interactive systems specifications for executability and prototypability. 7th Eurographics workshop on Design, Specification and Verification of Interactive Systems, DSV-IS'2000; LNCS n° 1946.
15. Olsen, D. R. 1998. Developing User Interfaces, Morgan Kaufmann.
16. Onizuka, M., Chan, F. Y., Michigami, R., and Honishi, T. 2005. Incremental maintenance for materialized XPath/XSLT views. In Proc. of WWW '05. ACM Press, 671-681.
17. Palanque P., R. Bastide. Petri nets with objects for specification, design and validation of user-driven interfaces. In proc. of IFIP Interact'90. Cambridge 27-31 August 1990 (UK).
18. Samet H., 1990, Applications of Spatial Data Structures: Computer Graphics, Image Processing, GIS, Addison-Wesley, Reading, MA, 1990.
19. Scalable Vector Graphics (SVG) 1.1 Specification http://www.w3.org/TR/SVG11/
20. Shan, Y. 1989. An event-driven model-view-controller framework for Smalltalk. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications. OOPSLA '89. ACM Press, New York, NY, 347-352.
21. Souchon, N., Vanderdonckt, J., A Review of XML-Compliant User Interface Description Languages, Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4-6 June 2003, LNCS, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 377-391.
22. Villard, L. and Layaïda, N. 2002. An incremental XSLT transformation processor for XML document manipulation. In Proc. of WWW '02. ACM Press, pp 474-485.
23. XSL Transformations (XSLT) Version 1.0 W3C Recommendation http://www.w3.org/TR/xslt

# Improving Usability of Interactive Graphics Specification and Implementation with Picking Views and Inverse Transformation

Stéphane Conversy
University of Toulouse – ENAC - IRIT
Toulouse, France
stephane.conversy@enac.fr

***Abstract*—Specifying and programming graphical interactions are difficult tasks, notably because designers have difficulties expressing the dynamics of the interaction. This paper shows how a specific architecture improves the usability of the specification and the implementation of graphical interaction. The architecture is based on the use of picking views and inverse transforms from the graphics to the data. With three examples of graphical interaction, I show how to specify and implement them with the architecture and how this improves programming usability. Moreover, I show that it enables implementing graphical interaction without a scene graph. This kind of code helps prevent errors due to cache consistency management.**

***Keywords-Usability of programming, Graphical Interaction, Specification, Implementation, Picking views, Inverse Transforms***

## I. INTRODUCTION

Interactive system programming is difficult, notably because designers have difficulties expressing the dynamics of the interaction [1]. Even if interaction is inherently graphical, specifying it and implementing it still relies mainly on textual languages that enlarge the gap between the phenomenon to describe and the description. Furthermore, writing interactive code with calculus-oriented languages is not suitable for describing reactive processes [2][3]. This results in so-called spaghetti [2] code that prevents readability and favors bugs, notably when the system grows after several increments. Finally, the need to make systems as fast as required by the interaction loop (short duration between user action/machine reaction/user perception) forces the designers to optimize their code and thereby make it difficult to read and modify.

I think that these problems pertain to the usability of specification and implementation of interactive graphics. Specifying interaction (referred to as "designing" in [1]) consists in describing how graphical representations react to user input. This is a problem that has been approached before with various languages (including visual), but as noted in [1], further work needs to be done to facilitate this task. Implementation is the process by which a programmer can turn a specification into executable code. Again, various approaches aimed at improving the transition and the readability of interaction code. Still, I think that a number of unimportant considerations hinder code readability and that a better architecture is necessary.

In this work, I rely on a particular architecture to ease specifying interactive graphics and to ease implementation of interactive graphics. The specification narrows the gap between the phenomenon and its description. The implementation paradigm enables the designer to use a data-flow architecture, which is more readable and more manageable than imperative code. I first present the architecture on which this work relies. After discussing a number of dimensions of analysis, I then present three examples of interactive graphics and argue that their specification and their implementation is more readable and understandable.

## II. MDPC

This section briefly introduces MDPC, the architecture I used (more details are available in [4]). MDPC is a refinement of the MVC architecture ('M' and 'C' denote 'Model' and 'Controller', the 'View' becomes 'Display view' and a 'Picking view' component is added). MDPC relies on two principles: "picking views" and "graphical transformations".

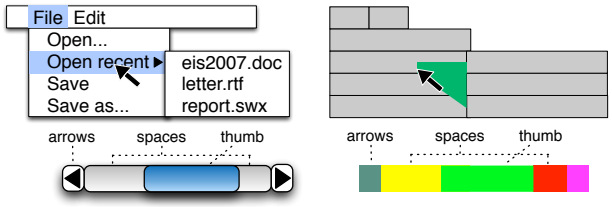



Figure 1. Top: display (left) and picking (right) view of a menu. Bottom: display (left) and picking (right) view of a horizontal scrollbar

Picking views are invisible graphical objects overlaid on visible ones, but that still react to user events. Fig. 01 shows the "display view" of a hierarchical menu (top left) and the corresponding picking view when the user is navigating in the menu (top right). The (transient) triangle laid over the menu in the picking view enables reaching the sub-menu entries while avoiding submenu folding. Similarly, the picking view of the scrollbar displays as many shapes as spatial modes (thumb, arrows and spaces between thumb and arrows). Picking views have two benefits. First, they help managing the dynamic of the states of the interaction (e.g. the transient triangle), as opposed to the graphical state of the display. Second, they enable to avoid analytical computation of spatial relationships (e.g. the movement with a direction below 45°, or the position of a click

with respect to the thumb) by using Enter/Leave events generated by the underlying graphical toolkit. Picking views actually reify *spatial modes* of interaction. A spatial mode is the spatial equivalent of a temporal mode: different behavior in function of space, versus different behavior in function of time.

Graphical transformations are functions that transform the conceptual model into graphics. MDPC uses two graphical transformations: one for the display view and one for the picking view. Fig. 02 shows the affine transforms applied to the model of a horizontal scrollbar (two values between 0 and 1) to generate the display view and the picking view. Computing the inverse transformations enable translating a graphical interaction (say a drag of the thumb) into operations on the model (translation of values).

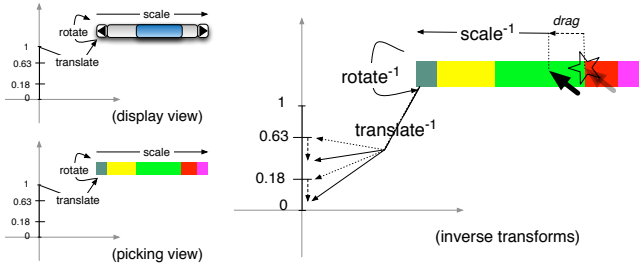



Figure 2. The graphical transforms (left) from the model to the views and the inverse transforms from the picking view to the model (bottom right).

MVC was the result of the application of the separation of concerns principle on interactive code [5] to improve modularity. Still, the MVC controller is in charge multiple concerns including the management of interactive state and the translation of events into operations on the Model. MDPC can be considered as the application of separation of concerns down to the MVC Controller itself. By using picking views and inverse transformations, MDPC offloads those two concerns from the MVC Controller. This makes the Controller code much simpler, almost eliminates the apparent impossibility to decouple the Controller and the View and makes Views and Controllers invariant from geometrical and layout transforms. This also improves modularity since the Controller can be made more general and reusable. For example, the same Controller can be used for various species of scrollbar (e.g. arrows at both ends, at one end, at the thumb ends; horizontal, vertical and radial layout). MDPC has been shown to make possible entirely "model-driven" implementation of scrollbars, sliders, range-sliders and hierarchical menus.

## III. DIMENSIONS OF ANALYSIS

I think that MDPC is also beneficial to the specification of interactive graphics and to their implementation. More precisely, using MDPC as a pattern helps at both designing the specification and designing the code. As such, MDPC can be considered as a method that improves the usability of programming.

Usability of "programming" (taken in a broad sense, i.e including specification and implementation) is the extent to which an environment (including language, pattern, IDEs etc) can be used to achieve programming tasks with effectiveness, efficiency and satisfaction (see [6] for an introduction). Usability is difficult to assess, because it requires longitudinal studies with a large number of designers (as defined in [1]). Since I have not done such studies in this work, I provide predictive evaluation of specification usability and implementation usability along three properties.

The first property that I assess is the descriptive power. i.e. the extent to which a designer using MDPC is able to specify and implement existing graphical interactions. This is a prerequisite for designers if I want them to be effective: they will not be able to design an intended interaction if the architecture does not allow for it. In the next section, I present three examples of specification and implementation of interactive graphics: Drag'n'Drop with hysteresis (direct manipulation technique [7]), Magnetic guides (instrumental interaction technique [8]) and a Calendar (complex representation combined with direct manipulation). Together, those examples aimed at showing that MDPC expressive power is sufficient to specify a large range of graphical interactions. In addition, I describe two kinds of implementation, one based on a scene-graph (Drag'n'Drop, Magnetic guides) and the other one based on a data-flow (Calendar). I show code snippets to help explain the implementation to the readers, to convince them that the implementation actually exists and runs and to enable them to replicate this work.

The second property that I assess is simplicity of description. Even if MDPC has a sufficient descriptive power, it would be useless if the description itself were cumbersome to specify and program. I provide an evaluation of simplicity of description by using concepts from the Cognitive Dimensions of Notation framework (CDN) [9] and from a list of desirable properties employed in the literature (see [3] for a survey).

The third property that I assess is the performance (implementation only). I also discuss this aspect since however elegant an implementation is, its usefulness can be reduced if performances are too weak.

## IV. DRAG'N'DROP WITH HYSTERESIS

The first example is the Drag'n'Drop with hysteresis, a direct manipulation technique. Drag'n'Drop with hysteresis forces the user to move past a small minimum distance from the ButtonPress position, before effectively triggering the Drag operation. This prevents the system from misinterpreting a Selection for a Drag'n'Drop: when selecting a graphical object with a click (ButtonPress then ButtonRelease), one or a few « Move » events may occur between the button events, because the mouse slips due to the force applied on the button by the finger. This makes the system misinterpret a Selection for a Drag'n'Drop and moves the selected object by a slight, but undesirable amount.

### *A. Interaction specification*

A traditional analytical algorithm consists in computing at each Move event the distance between the ButtonPress position and the cursor position, testing if the distance is superior to the minimum distance and moving the object if the test is successful. This necessitates the computation of a Euclidean distance (square root of sum of squares).

*1) Description*

The version with MDPC consists in drawing an invisible circle centered on the position of the ButtonPress, with a radius equal to the hysteresis distance. Fig. 03 shows the display and picking views for explanation purpose: the circle is visible, but in the real system it is not. At the beginning, the cursor is at the centre of the circle. If the cursor does not leave the circle before a ButtonRelease, the interaction is interpreted as a Select. If the cursor leaves the circle, the minimum distance is reached and the Drag can start. The invisible circle is removed, which allows the user to move the object within a distance from its initial position smaller than the hysteresis distance.

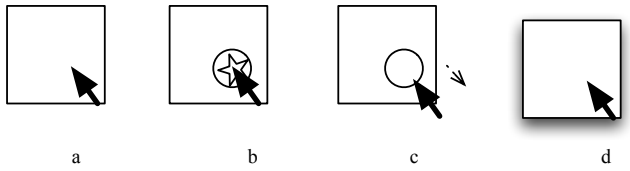



Figure 3. Hysteresis with MDPC. (a) hover (b) press:an invisible circle is inserted into the scene (visible here for explanation) (c) no drag while the cursor stays in the circle (d) leaving circle: removal of the circle, drag starts.

*2) Simplicity*

I think that the MDPC description is closer to the conceptual model of the interaction. In fact, computing the distance between the cursor and its initial position at each Move event is not necessary for specifying the interaction. The only information needed is the minimum distance to be reached. Since this distance is reified into a circle, the concept of distance crossing is more directly represented. Hence, MDPC improves the Closeness of Mapping cognitive dimension. Finally, the designer can make picking view visible for debugging purpose. By directly seeing the circle on the screen, one can understand how the graphical interactive state behaves and debugs more easily than with code only. Here, MDPC improves the Visibility cognitive dimension.

## *B. Implementation*

This particular implementation uses the SwingStates toolkit [10]. SwingStates enables programming interaction with state machines directly in java files. The transition between states can be guarded (i.e. a predicate prevents the transition to fire) and can trigger an action when fired. SwingStates relies on a scene graph, i.e. a data structure that retains graphical objects. With SwingStates scene graph, graphical objects may be "tagged": any operation on an object can also be applied on a "tag", meaning that any object with this tag will be modified accordingly. The following code heavily used this feature.

*1) Description*

The state machine is shown in Fig. 04. When the user presses on an object, the current state becomes "waitHyst" and waits for the hysteresis distance to be crossed. The code of the action associated to the "Press" transition is shown in Fig. 05. The picking shape is created (CEllipse, the circle), made invisible, then added to the scene graph. Graphical objects of the picking view are invisible to the user, but react to mouse events. As said before, one can comment the line that make objects invisible for debugging purposes.

The circle is centered at the location of the cursor: hence, the cursor is inside the circle. The "leave" transition pertains to this circle (code not shown in the figure): when the cursor leaves the circle, the "leave" transition to the "dragging" state is fired, an action removes the invisible circle from the scene graph and the user is free to drag the object around.

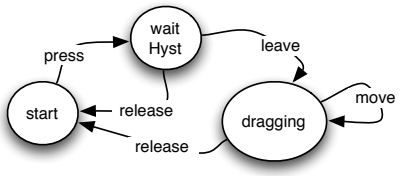



Figure 4. Hysteresis state machine. Circles denote states, arrows transition. The text on a transition denotes the interaction event that triggers a transition.

```
public State start = new State() {
  Transition press = new PressOnShape(BUTTON1, ">> waitHyst") {
    public void action() {
      toMove = getShape(); // get the object to drag
      lastPoint = getPoint(); // store last clic position
      hystShape = new CEllipse(lastPoint.getX()-5,lastPoint.getY()-5, 10,
10); // picking shape
      hystShape.setDrawable(false); // set invisible
      canvas.addShape(hystShape); // add to scene graph
    }};};
```

Figure 5. Action on "press" transition from "start" to "waitHyst".

*2) Simplicity*

Even if simple, the MDPC-based description of the interaction illustrates how Enter and Leave events are used in place of analytical computation of the Euclidean distance. Hence, the designer is not required to write this code. Of course, with the traditional way, one could have used abstraction and call a 'distance' function instead of writing the distance code, but the MDPC version gets rid of this necessity.

This example also illustrates how picking views help manage the dynamics of the interaction state. Finite State Machines are well adapted to MDPC descriptions of the interaction. At each state can correspond a particular picking view, which is active when the state is active. This is similar to the architecture described in [11]. Again, MDPC improves Closeness of Mapping with interactive state implementation.

*3) Performance*

Adding a single circle to a scene graph is inexpensive. The generation of Leave/Enter events may actually use a Euclidean distance, hence the computation is the same as the traditional algorithm.

# V. MAGNETIC GUIDES

Magnetic guides are instruments for aligning graphical objects [8]. During the Drag'n'Drop of an object, if the object is close enough to the magnetic guide, the guide attracts the object: hence, dropping multiple objects on a linear guide makes them aligned. More complex alignments allow for alignment of objects center, but also of their boundaries. More complex guides include Bezier curves. Alignment with

magnetic guides is an example of instrumental interaction [8]: a Magnetic Guide is an instance of an instrument i.e. action (alignment) reified into an interactive object that control other interactive objects. Magnetic guides are different from a "grid", since they are explicitly defined and manipulated by the user.

### A. *Interaction specification*

As in to the previous example, a traditional analytical algorithm computes the distance between the guides and the dragged object, tests if the distance is inferior to the attraction distance and sticks the object on the guide if so.

#### 1) *Description*

Fig. 06 shows both the display view (dashed line and green rectangle) and the picking view (gray rectangles, red square) for illustration purpose. With the MDPC pattern, the algorithm consists in drawing an invisible thick line over the guide (thin dashed line on the figure), whose thickness is equal to two times the attraction distance (Fig. 06, gray rectangles), and in registering a callback when the cursor enters or leaves the invisible thick line (events "Enter" and "Leave"). Thus, when the cursor enters the invisible thick line, the object sticks to the guide; when the cursor leaves the thick line, the object sticks to (and thus follows) the cursor.

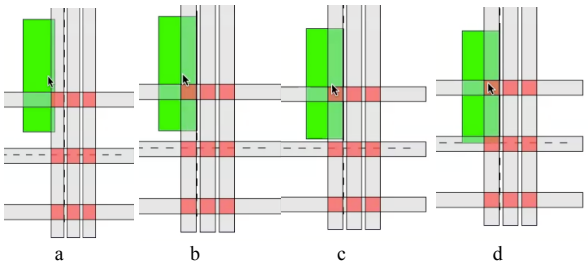



Figure 6. Dashed lines: magnetic guides; gray rectangles : picking view of magnetic zones ; red squares: picking view of magnetic zones shared by two guides (a) free drag (b) right horizontal alignment (c) just before entering in the magnetic zone to align vertically, at the bottom (d)

As said earlier, more complex guides allow for alignment with the center of objects, but also with their boundaries. With MDPC, this is described with multiple picking zones, placed around the magnetic guides with respect to the geometry of the object and the position of the cursor relative to the object (Fig. 06, gray rectangles). In addition, guides may intersect, allowing an object to stick to their intersection and keep alignment with two sets of objects. Drawing two thick lines results in a partial occlusion of one line by the other at the intersection point. With a toolkit that can synthesize Enter and Leave events for occluded objects, no adaptation of the previous algorithm is necessary. However, with the SwingStates' event synthesis model, the previous method does not work: the topmost guide would prevent the attraction from the occluded line since no Enter or Leave event would be emitted for the occluded thick line. With such a model of events, it is necessary to define the area of intersection between thick lines and make the object stick at the intersection when the cursor is in the intersection area (Fig. 06, red squares).

#### 2) *Simplicity*

The interaction is complex, and the distances to compute are numerous: there are 6 distances per guide (3 vertical, 3 horizontal) and the reference point from which to compute the distance is not easy to grasp and understand. MDPC encourages the identification of spatial modes of interaction and their corresponding area. I think that thinking in terms of area of attraction is easier. As noted in [1], designers often use drawings to explore a solution and explain them to colleagues. MDPC allows designers to use these drawings directly to express the interaction. In addition, when the guide themselves are complex (e.g. curves), no additional cost in terms of reasoning is necessary compared to the distance model. Similarly to the Drag'n'Drop example, MDPC thus improves Closeness of Mapping and Visibility.

The intersection area problem induces more coding for the designer than the distance computation model. The MDPC solution seems more complex than computing distances from guide: the burden of describing intersection shapes may not make MDPC as advantageous as claimed. This hinders the Terseness cognitive dimension. It must be underscored however that this problem only occurs with scene graphs that do not generate Enter/Leave events for occluded objects.

### B. *Implementation*

#### 1) *Description*

This implementation also uses the SwingStates toolkit. The state machine is shown in Fig. 07. The interaction begins with the hysteresis interaction described earlier. When crossing the hysteresis distance, the "leave" transition is fired and the machine enters the "dragging" state.

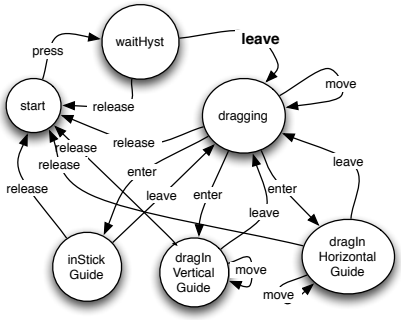



Figure 7. Magnetic guide state machine

Picking views are managed in the code of the action associated to the "leave" transition (shown in bold in Fig. 07). The code itself is shown in Fig. 08. First the previous picking views (hysteresis circle) is removed (a) and replaced by three picking objects per guide (b), to align with the center and the boundaries of the dragged object. At the beginning, the picking objects are put on the position on the guide. The objects for the boundaries are then "spread around" the guide by a distance equal to half the height or width of the dragged object (c). Then, all guides are moved by a distance equal to the shift between the position of the cursor inside the dragged object and its boundaries (d). When the cursor enters the picking shape of a guide, the machine enters the corresponding state. In the "dragIn*Guide" state, the move transition triggers an action

that moves the object along the guide. In “inStickGuide”, no action (and thus transition) is necessary on a “move”.

```
public State waitHyst = new State() {
  Transition drag = new LeaveOnShape(">> dragging") {
    public void action() {
      […]
      // (a) remove previous picking view
      canvas.removeShape(hystShape);
      // (b) create horizontal pick shapes
      for (int i=0; i<3; ++i) {
        CShape s = new CShape(new
BasicStroke(20).createStrokedShape(new Line2D.Double(0, 0, 500, 0)));
        canvas.addShape(s);
        s.addTag(hMagnetTag);
        // (c) spread the pick shapes around the guideline
        if (i==0) s.translateBy(0,toMove.getHeight()/2);
        if (i==2) s.translateBy(0,-toMove.getHeight()/2);
      }
      // translate around guideline
      hMagnetTag.translateBy(0,ymg);
      // (d) translate the pick shapes according to the relative position of
the cursor from the reference point of the shape (middle)
      hMagnetTag.translateBy(0,pickRelPos.getY());

      // create vertical pick shapes and sticky pick shapes at h and v
intersections
      // hidden: similar to horizontal guides
}}}
```

Figure 8. Action on “move” transition from “waitHyst” to “dragging”

*2) Simplicity*

For simple guides, such as horizontal or vertical guides, the computation of the position of the dragged object stuck to the guide is straightforward: one of the Cartesian dimensions is that of the cursor and the other is that of the guide. In the case of a more complex guide, it is necessary to code the computation of the orthogonal projection of position of the dragged object on the guide and sets its coordinates to the coordinates of the projection.

Since SwingStates does not synthesize Enter and Leave events for occluded objects, the code has to create the picking objects for the intersections. In the simple case of horizontal and vertical guides, the shape of the intersection is a square centered at the intersection of the guides. However, more complex guides may require more complex computation. In this case, MDPC extends nicely to the use of the AND operation of the constructive area geometry and the computation the shape resulting from an AND between the two thick lines. Some toolkits provide such algorithms (i.e. Java2D Shape API, or OpenGL GLU tesselator [12]).

*3) Performance*

Again, in order to make reasoning easier, the code avoids analytical computation by relying on the algorithms provided by the scene graph. The test for shape inclusion does not require a rasterization. Instead, the algorithm in the scene graph may use the distance algorithm that one would have used in the interaction code. Hence performances are similar.

With SwingStates model of events, additional computations of area are necessary. However, those computations happen only once during the interaction (in the transition between “waitHyst” and “Dragging”).

## VI. CALENDAR

The next example is a Calendar application, with a “week” view on events, such as Apple’s iCal or Google Agenda. Fig. 09 shows the overall display (top) and picking (bottom) view. I have replicated two interactions: “Drag’n’Drop” of calendar entries, which allows the user to move an entry in the day, or to move it into another day of the displayed week; and the “Resize” of the duration of calendar entries.

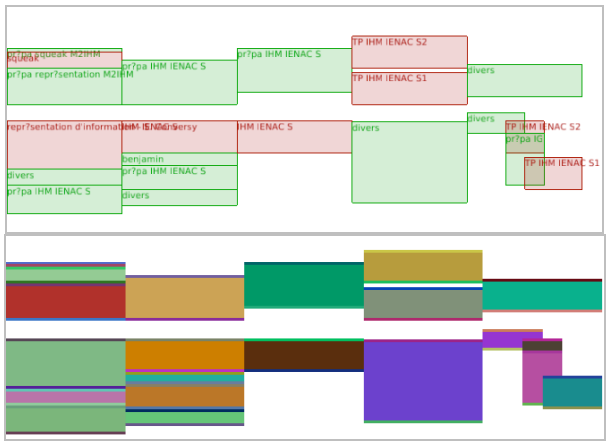



Figure 9. The “display” view (top) and the corresponding “picking” view (bottom) of a calendar. The picking algorithm uses unique colors for each picking object, which explains the colorful picking view.

### A. Interaction specification

A traditional algorithm uses the positions and analytical distance computation to decide the reaction to user events.

*1) Description*

With MDPC, the “Display” view of each calendar entry is a rectangle (Fig. 10). The top edge reflects the date and time when the entry starts, while the bottom edge reflects the date and time when the entry ends. The width of the entry is not tied to the data: it is equal to the width of a column, in this case a seventh of the window since a week contains seven day. When multiple calendar events overlap, the corresponding rectangles share the column width (left most column in Fig. 09).

The picking view of each entry is composed of three juxtaposed rectangles (Fig. 10). The middle rectangle is similar to the rectangle of the display view and its height depends on the entry duration. A Drag’n’Drop of this rectangle allows modifying both the start and end time without modifying its duration. The two other rectangles allow the user to pick the top (resp. bottom) edge of an entry and change the start (resp. end) of the entry by direct manipulation. The modification of the data is done thanks to an inverse transformation, as explained in the next section.

*2) Simplicity*

The gain in simplicity is the same as in the previous examples: this improves Closeness of Mapping and Visibility.

### B. Implementation

The previous examples use Java and a scene-graph. They illustrate the use of picking views for managing interaction

state and for avoiding analytical computations. I implemented the calendar example with Tcl [13] and OpenGL [12], and by relying on a data-flow. This demonstrates not only the use of picking views, but also the use of inverse transformations, the second principle of MDPC. It also shows that MDPC is independent from the language and does not require a scene-graph.

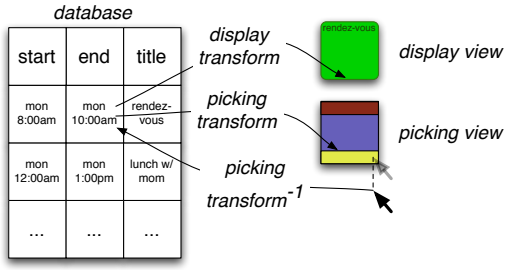



Figure 10. Display and Picking view of a calendar entry. The position of the cursor is transformed back into the conceptual model by using the inverse picking transformation.

### 1) Description

The architecture is shown in Fig. 10). Calendar entries are stored in a relational database table. The table includes a "start", an "end", and a "title" column. A SQL select allows selecting visible entries and computing the value needed for the visualization. Each frame rendering triggers two OpenGL-based redisplay functions, one for the display view (proc view, display view, Fig. 11) and one for the picking view (proc view, picking view, Fig. 11). The display transformation fills pixels in the frame buffer, while the picking transformation fills pixels in an offscreen buffer. Both transformations share a *transf* function (Fig. 11, middle-left). *transf* first wraps the data multiple times on X and Y (Fig. 12). The *wrap* function (shown in Fig. 11, bottom-left) is more complex than necessary (since I only use the week view), but serves as a demonstration that even a complex function can be reversed. Once wrapping is done, the position in the day is computed and displayed on the screen's Y dimension quantitatively (yInDay). This leads to a 2-D position expressed in terms of cells (e.g. (3; 4.5)), which is then multiplied by the actual display size of a cell (CellWidth x CellHeight). Finally, the *transf* function applies a user-controlled pan and zoom. A final computation shifts the x position of events inside a cell to take into account parallel entries (Fig. 09, right).

The code that manages user input is shown in Fig. 11, right. When the user presses on and moves one of the small rectangles in the picking view of a calendar entry, an inverse transformation is applied on the X and Y dimensions of the Move event. Since the position of the rectangles is the result of the application of a continuous and monotonous function on a scalar (a time), it is sufficient to apply the inverse function to the position of the cursor to get the corresponding value in the referential of the data model. The inverse *transf* is shown in Fig. 11, middle-right and the inverse *wrap* is shown in Fig. 11, bottom-right. Finally, a SQL query update modifies the data in the data table. After each modification (hence each movement), the system triggers a redisplay and the modification is visible immediately.


Figure 11. Actual code for calendar. Left: disp. & pick. views, transformation (transf) and wrapping (wrap) - Right: their inverse (pick, invtransf, invwrap). Note the symmetry or anti-symmetry of functions and their inverse.

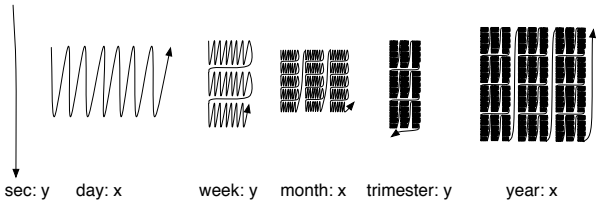



Figure 12. A calendar is a wrapped view of time over X and Y

### 2) Simplicity

The display is the result of the application of a function on the data. The first advantage is that the understanding of how the model is transformed on the screen is easier to grasp, because it only depends on an identified flow and is not spread around the entire program (Fig. 11): in other words, spaghettis untangle [2]. This improves Locality [3] and thus Visibility. The second advantage is that if the function is a reversible transformation (which is the case here), the design of the function that transforms user manipulations into results on the model is straightforward: it consists in applying inverse sub-functions in reverse order. Moreover, the visualization of the program text helps to design such an inverse function, because of the Symmetry [3] between transformation and their inverse (Fig. 11). When designing the display and the interaction, a good way for a designer to get confidence in the code is to

target and reach this symmetry and verify that for each sub-function there is an inverse sub-function.

Using functional code enables the implementation to use a data-flow. When applying modifications to the model, all depending variables (in particular all graphical positioning properties) are recomputed and displayed immediately. There is no need to manage consistency, which reduces the Viscosity cognitive dimension. Variables external to the model also benefit from data-flow. For example, the width and height of a cell depend on the containing window. When the user resizes the window, the size of cells adapts "automatically".

*3) Performance*

If it is simpler to manage than analytical computation, this architecture is more costly in terms of computation. For example, it is necessary to recompute for each modification the tessellation and the rasterization of each graphical object. This behavior is similar to 3D applications and games: with 3D scenes, since the point of view may differ for each frame, coders do not bother implementing algorithms that manage damaged zones and usually redisplay all objects. I think that, given the computing power available since the advent of 3D games, it is more beneficial to trade performances for ease of coding. Besides, the description with a data flow can help optimizing performances: it is possible to consider the chain of transformation from data to pixels as a compiler and use automatic optimization provided by a graphical compiler [14] (partial evaluation, automatic cache, dead-code elimination, etc). Finally, if a data-flow may be more costly in terms of computation, it is less costly in terms of memory since it does not retain graphics.

## VII. DISCUSSION

This section synthesizes the benefits of using MDPC for specification and implementation.

### *A. Software Engineering*

As explained in [4], MDPC improves software modularity. The role of the Controller of MDPC is limited to the management of the dynamics of the interaction state. In the Drag'n'Drop and Magnetic guide, the controller is reduced to the state-machine. In the Calendar example, the Controller is the interaction code. Since the Controller is independent from geometrical or layout transforms, it can be reused across multiple interactions. For example, if a pan is applied to the D'n'D or Magnetic guide scenes, there is no need to change the interaction code. This is particularly visible in the Calendar example: the same code can be used regardless of the fact that pan and zoom is handled by the application. One can add a rotation at the end of the *transf* function and its inverse at the beginning of the *invtransf* function (for example to implement interactions from [15]), with no need to modify further the existing code. The interaction of the user will still be perfectly transformed into operations on the model.

It is important to note that it is the combination of picking views and inverse transformations that enables this feature. Using picking views radically simplifies the code and cancels the need for complex adaptation of analytical code when one adds a new transformation. And transformations are an abstraction which is both independent from the notion of interactive state and can still be applied easily to the reification of interactive state into picking views.

### *B. Implementation: scene graph considered harmful*

The implementation of the calendar uses a paradigm that contrasts with the paradigm relying on a scene graph. Often, implementers use a scene-graph to retain the properties of the graphical objects and to optimize the rendering. In fact, a scene graph is also a "cache" of the rendering pass. As a cache of graphical properties, it relieves the designer from the apparent obligation to retain the graphical objects for subsequent redisplay. As a cache of transforms, it optimizes the redisplay: often, the modification between two frames is minor and one can expect better performance if previous computation is reused.

However, as with any "cache", consistency must be dealt with. Consistency management is known to be error-prone and even if it seems compulsory to users of scene-graph, it requires caution to be taken, hence time and resources, at the expense of other concerns. I think that graphics management, user input management and data update are hindered by consistency management. The data-flow architecture inherently eliminates cache management problems, since there is no cache anymore. Thus, getting rid of scene-graph makes implementing interactive graphics easier.

Two arguments may counter this claim: performance and lack of services of scene-graph-less code. As for performance of data-flow, I have already noticed that highly demanding 3D applications behave this way and are efficient. Furthermore, some interaction requires drawing the entire scene. For example, resizing the window of the calendar application leads to a complete computation of all graphical elements in the scene. In this case, the advantage of the scene graph is null, since it does not act as a cache anymore (the cache is invalidated at each rendering pass). Besides, the use of a graphical compiler offloads optimization concerns from the programmer to a tool [14].

As for services, a number of them provided by a scene graph (ready-to-use graphical shape rendering, picking management) do not require a data structure that retains graphics. For example, the graphical properties need not be retained, since the transformations that lead to those graphical properties are *retained in the code*: graphical properties can be generated at each redisplay. In the same way, picking does not require a complex scene-graph. In the calendar example, picking is realized with a "pick by color" algorithm [16].

## VIII. RELATED WORK

A number of works have tackled usability of programming, including psychology of programming, cognitive dimensions of notation [9], or API usability [17]. For example, [10] and [11] enable the programmer to describe interactive state with state machines [18]. Most usability studies target general-purpose languages or APIs rather than tools for building interactive systems [3]. Exceptions include Myers' study of the programming practices of graphical designers [1]. Our work builds on these concerns and proposes a practical method that

aims at improving usability of specification and implementation of graphical interaction. Artistic resizing is a technique that enables to specify how graphical components resize when users resize the container window [19]. It is an example of how specification can be turned from a program into graphical description. Our work pursues this effort, in that it improves the Closeness of Mapping between the phenomenon and its description.

Describing graphics with Data Flow has been extensively studied in the past. For example, Fabrik is a direct manipulation-based user interface builder that enables a designer to specify transforms between widget with a visual flow language [20]. Events flow in the same flow graph that describes the geometrical transforms, so that they are automatically transformed to a position relative to the graphically transformed widget. Garnet uses one-way constraints, which can be considered as data flow to propagate changes [21]. In order to improve interactive graphics programming, [22] proposes solutions to facilitate mixing of data flow of input and scene graph for output.

The inverse of model-view matrix is often used to retrieve an object that has undergone multiple 3D transforms (due to a change of point of view, or due to modeling) [12]. [23] discusses how to enable users to change data through visualization and a data-flow. Metisse [15] and Façade [24] rely on inverse transforms to handle user manipulation in rotated views. However, none discusses how to design inverse transformations to reflect users' manipulation into the models.

## IX. CONCLUSION

I have presented how the MDPC pattern - based on picking views and inverse transformations - can facilitate specifying and implementing graphical interaction. I have evaluated positively its ability to describing a large range of graphical interaction. I have also assessed the simplicity of description by identifying the benefits (modularity, closeness of mapping, visibility, locality and symmetry of code). Of course, there are some drawbacks (terseness and performances in certain cases) and the claims, even if supported analytically, must be experimentally tested. Furthermore, I do not claim that MDPC is adapted to all graphical interaction. For example, one would better apply a modulo operation to the cursor position to align objects on a grid, instead of relying on one picking shape per row or column on the grid. However, I believe that thinking in terms of reified spatial modes of interaction and transformations facilitate designing an interaction. In the future, I plan to separate even further the implementation of graphics and the implementation of transformation by using specialized languages (e.g. SVG as in [14]) and to explore optimization and especially cache management.


## ACKNOWLEDGMENT

S. Chatty, H. Gaspard-Boulinc, Y. Jestin, C. Letondal and B. Tissoires help improve the paper, many thanks to them.


## REFERENCES


[1] B. Myers, S. Park, Y. Nakano, G. Mueller and A. Ko, "How designers design and program interactive behaviors," in *Proc. of IEEE VL/HCC*, pp. 177-184, 2008.

[2] B. A. Myers, "Separating application code from toolkits: eliminating the spaghetti of callbacks," in *Proc. of ACM UIST*, pp. 211–220, 1991.

[3] C. Letondal, S. Chatty, G. Phillips, F. André and S. Conversy, "Usability requirements for interaction-oriented development tools," in *Proc. of PPIG*, 2010.

[4] S. Conversy, E. Barboni, D. Navarre and P. Palanque, "Improving modularity of interactive software with the MDPC architecture," in *Proc. of IFIP EIS*, pp. 321–338, 2007.

[5] T. Reenskaug, *Models - views - controllers*. Xero PARC, 1979.

[6] T.R.G. Green and M. Petre. "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework," in Journal of Visual Languages and Computing, (1996), p131-174.

[7] B. Shneiderman, "Direct Manipulation: A Step Beyond Programming Languages," *Computer*, vol. 16, no. 8, pp. 57-69, 1983.

[8] M. Beaudouin-Lafon, "Instrumental interaction: an interaction model for designing post-WIMP user interfaces," in *Proc. of ACM CHI*, pp. 446–453, 2000.

[9] T. Green, "Cognitive dimensions of notations," in *Proc. of HCI*, p 443-460, 1989.

[10] C. Appert and M. Beaudouin-Lafon, "SwingStates: adding state machines to Java and the Swing toolkit," *Software—Practice & Experience*, vol. 38, pp. 1149–1182, Sep. 2008.

[11] S. Chatty, S. Sire, J. Vinot, P. Lecoanet, A. Lemort and C. Mertz, "Revisiting visual interface programming: creating GUI tools for designers and programmers," in *Proc. of ACM UIST*, pp. 267–276, 2004.

[12] M. Woo, J. Neider and T. Davis, *Opengl Programming Guide: The Official Guide to Learning Opengl, Version 1.1*, Addison-Wesley, 1997.

[13] J. Ousterhout, "Scripting: higher level programming for the 21st Century," *Computer*, vol. 31, no. 3, pp. 23-30, 1998.

[14] B. Tissoires and S. Conversy, "Graphic rendering considered as a compilation chain," in *Proc. of DSVIS*, pp. 267–280, 2008.

[15] O. Chapuis and N. Roussel, "Metisse is not a 3D desktop!," in *Proc. of ACM UIST*, pp. 13–22, 2005.

[16] P. Hanrahan and P. Haeberli, "Direct WYSIWYG painting and texturing on 3D shapes," in *ACM SIGGRAPH Comp. Graphics*, p215–223, 1990.

[17] B. Myers, *Usability issues in programming languages*. School of Comuter Science, CMU, 2000.

[18] R. J. K. Jacob, "A Visual Language for Non-WIMP User Interfaces," in *Proc. of Visual Languages*, pp. 231–, 1996.

[19] P. Dragicevic, S. Chatty, D. Thevenin and J. Vinot, "Artistic resizing: a technique for rich scale-sensitive vector graphics," in *ACM SIGGRAPH 2006 Sketches*, 2006.

[20] D. Ingalls, S. Wallace, Y. Chow, F. Ludolph and K. Doyle, "Fabrik: a visual programming environment," in *ACM SIGPLAN Notices*, pp. 176–190, 1988.

[21] B. T. Vander Zanden et al., "Lessons learned about one-way, dataflow constraints in the Garnet and Amulet graphical toolkits," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 23, pp. 776–796, Nov. 2001.

[22] C. Appert, S. Huot, P. Dragicevic and M. Beaudouin-Lafon, "FlowStates: prototypage d'applications interactives avec des flots de données et des machines à états," in *Proc. of IHM*, pp. 119–128, 2009.

[23] T. Baudel, "From information visualization to direct manipulation: extending a generic visualization framework for the interactive editing of large datasets," in *Proc. of ACM UIST*, pp. 67–76, 2006.

[24] W. Stuerzlinger, O. Chapuis, D. Phillips and N. Roussel, "User interface façades: towards fully adaptable user interfaces," in *Proc. of ACM UIST*, pp. 309–318, 2006.

# Hayaku: Designing and Optimizing Finely Tuned and Portable Interactive Graphics with a Graphical Compiler

**Benjamin Tissoires** [1,2]
[1] DSNA DTI R&D
7, avenue Edouard Belin
31055, Toulouse,
France
tissoire@cena.fr

**Stéphane Conversy** [2]
[2] Université de Toulouse, ENAC, IRIT
7, avenue Edouard Belin
31055, Toulouse,
France
stephane.conversy@enac.fr

## ABSTRACT
Although reactive and graphically rich interfaces are now mainstream, their development is still a notoriously difficult task. This paper presents Hayaku, a toolset that supports designing finely tuned interactive graphics. With Hayaku, a designer can abstract graphics in a class, describe the connections between input and graphics through this class, and compile it into runnable code with a graphical compile chain. The benefits of this approach are multiple. First, the front-end of the compiler is a rich standard graphical language that designers can use with existing drawing tools. Second, manipulating a data flow and abstracting the low-level run-time through a front-end language makes the transformation from data to graphics easier for designers. Third, the graphical interaction code can be ported to other platforms with minimal changes, while benefiting from optimizations provided by the graphical compiler.

### Author Keywords
Human-Computer interfaces, User Interface Design, Methods and Applications, Optimization

### ACM Classification Keywords
H.5.2 User Interfaces: GUI.

### General Terms
Design, Languages

## INTRODUCTION
Interactive graphics development is a notoriously difficult task [18, 19]. In particular, rich interactive systems design requires finely-tuned interactive graphics [13], which consists of a mix of graphical design, animation design and interaction design. Subtle graphics, animations and feedback enhance both user performance and pleasure when interacting [16]. The success of the iPhone demonstrates it: finely tuned widgets, reactive behavior, and rich graphics together make the iPhone interface superior to other products. Designing such systems is a recent activity that has rarely been supported explicitly in the past. Yet, their quality is essential for usability. Unfortunately, developing such software is not reachable by all stakeholders of interactive system design. This requires highly trained specialists, especially when it comes to using very specific graphic concepts *and* optimize the rendering and interactive code. Hence, there is a clear need for making interactive graphical programming more usable.

Moreover, even within a given style of computing (either web or mobile), new means of thinking, designing and developing interfaces arise every couple of years. For example, we successively saw the rise of Java2D, Adobe Flash, Adobe Flex, Microsoft dot net, XAML, SVG, WMF, Web 2.0 interfaces programmed in javascript in the browser (with the Canvas and HTML5), OpenGL etc.[1]. In order to design and develop interactive systems on those platforms, interface designers have access to a plethora of toolkits, usually incompatible with one another. This results in the failure of *reusability*, one of the most praised property in computing: designers have to redevelop existing software in order to port it to another platform, with the associated drawback of not reusing well crafted and tested software. For example, the menu subsystems that have reached a good level of usability in traditional desktop platforms (i.e. Windows or MacOSX) are poorly imitated in Web 2.0 interfaces, where the user is for instance required to follow a tunnel strictly when navigating in a hierarchical menu. Hence, there is a need for the ability to reuse existing software, especially if we assume that new platforms will keep appearing in the future (see WebGL for example).

This paper addresses the two requirements presented above: design usability and reusability of finely-tuned interactive gra-phics. In particular, we introduce Hayaku, a toolset that targets interactive graphics that Brad Myers refers to as the "insides" of the application [21], and that no widget toolkit can support. After a review of related works, we present the exact audience that we target, and the requirements of such an audience. We then present the toolset using three use cases, and some of the internal mechanisms that implement


*EICS'11*, June 13–16, 2011, Pisa, Italy.


[1] . . . , Cairo, Qt, Prefuse, Protovis, iPhone SDK, Open Handset Alliance's Android, Palm WebOS to name a few more

its features. We finally provide a number of elements to evaluate the toolset according to our claims. Related Work This work is related to two topics in the user interaction software and technology community: methods to design interactive systems, and graphical toolkits.

#### Interactive System Design Methods

Chatty et al. [6] present a method and associated tools to involve graphic designers in interactive system design and development. Programmers and graphic designers first agree on a conceptual and simple SVG skeleton of the scene. While programmers code the interaction with a low quality representation, graphic designers can work on their design in parallel. Since programmers and designers respect a contract, the production of the final system consists in the replacement of the low quality representation by the designer's one. However, the tools oblige the graphic designer to use a library to transform the high-level language (SVG) to a lower-level one (a Tk-like canvas) with a lesser expressive power. This hinders exploration of alternative design since changing gra-phics implies many manipulations to reflect the change in the final application. Furthermore, when optimizing code, the approach falls back to a sequential process: programmers have to wait for designers' solutions before optimizing by hand the rendering code, and designers have to wait for optimizations to assess if their design is usable.

Microsoft Expression Blend makes heavy use of XAML to describe the graphical parts of the application. Like Intuikit, the aim is to separate the graphical description from the functional core of the application. The designer can produce one design per C# class that has to be drawn, but he still needs to manipulate the low level code in order to implement interactions and animations. The concept of "binding" allows programmers to link the graphical shapes and the source objects. The Adobe Flex and Flash suite also provides a means to separate the graphical description from the functional core. However, even if the designer can rely on Flash to build her graphical components, she has to develop the rest of the application using the ActionScript language. Furthermore, there is no abstraction of the graphics, nor a way to express properties with a data-flow. Finally, even if Flex runs on a variety of platforms in its own window, it is not possible to embed the graphics among the graphical scene of another application.

#### Toolkits

Many toolkits address the problem of performance: Prefuse [14], Jazz [4], Piccolo [3], and Infoviz Toolkit (IVTK) [9] for instance. Performance is maximized by using specialized data structures explicitly (tables for Prefuse and IVTK), or hidden data structures (spatial tree for Piccolo). The first limitation of this approach is that the language used to describe graphics is both inappropriate and not rich enough: describing graphics in Java code with SwingStates [1] is verbose, Java concepts do not match graphics exactly, and rich graphics created with tools for graphic design cannot be directly used in the toolkit. Rich graphics toolkits exist, such as Batik[2], but they are not efficient performance-wise. Furthermore, the problem with these toolkits is that even if they are efficient, they force the toolkit user to work with a specific language and a specific run-time. For instance, users of toolkits can not use Prefuse to write a C or C++ application.

Other works use compiler-like optimizations to produce efficient graphical code (Java3D, LLVM [15] with Gallium3D[3] in Mesa). However these tools are only accessible to low level graphical programmers that manage to write code for the graphic card directly. They are not supposed to be used by the average interactive application developer, with basic understanding of the factors that accelerate rendering.

The solution we propose here consists in helping the production of efficient code for heavy graphics handling. In order to compile the graphical part, we rely on a dataflow, and a mechanism that is able to track the dependencies between input data and graphical elements. Dataflow has been used in graphical interactive toolkits (Icon [8] for the input, and Garnet [25] for the constraints), and have been showed to help building interactive systems efficiently. However, the main difficulty with such a system is to make it fast for both graphical rendering *and* the dependency updating mechanism.

This point is addressed in [24], which introduces a compile chain for interactive graphical software. This work shows that using a graphical compiler (GrC) together with a dataflow leads to good performance. However, the tools were more a proof of concept than a real toolset: the authors present a way to implement optimizations, but do not detail how programmers of the graphical interface can connect all parts together. Another problem of the GrC is that it generates a program that is linked to the runtime of the GrC. This forces the designer to describe all graphical parts of the application with the GrC. However, when dealing with high performance applications, there are parts of the code that the programmer still wants to write manually, in order to maximize the performances. Meanwhile, this programmer may not want to write every piece of the software, if only because they already exist.

The work we present in this paper improves on the concepts described in [24]. In particular, we show how our new tool can help the programmer and the graphic designer to use graphical compiling in a simpler manner, and what benefits can be gained from the new functionalities. We also improve it by allowing it to be modular and thus producing embeddable components. Finally this modularity allows us to turn this proof of concept into a real compiler that can handle multiple graphics back ends and run-time modules.

## TARGET AUDIENCE AND REQUIREMENTS

The work presented here targets members of interactive system design teams. A large body of work aims at supporting interactive system design. For example, Participatory Design (PD) partly aims at facilitating production and communication between all designers, be they user-experience specialists, graphic designers, users, programmers [17]. PD employs multiple means to elicit design and communicate it

[2] http://xmlgraphics.apache.org/batik/

[3] http://wiki.freedesktop.org/wiki/Software/gallium

efficiently in groups where people do not share the same culture. Use cases in the form of stories, drawings and mock-up [5], paper prototypes [23]: all tools aim at maximizing expression, exploration by iteration and understanding by culturally different designers.

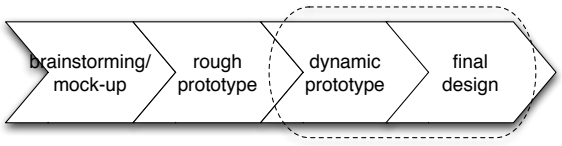



**Figure 1. Targeted activity**

After these tools have led to initial static prototypes, the designers have to work on dynamic, graphical interactive prototypes [2] [6] (figure 1). As said in the introduction, part of the work is the design of finely-tuned interactive graphics, which consists in a mix of graphical design, animation design and interaction design. The quality of the artifacts designed during this stage in the process is essential for usability. The overall user experience of interacting depends on how well all features (be they graphical, animation, behavior) mix together: the designer must address all concerns at the same time, and dispatching the task between a graphical designer and a programmer does not work anymore. Hence, this activity requires designers with skills in graphic design, animation, interaction design and programming. Our work especially targets this kind of designers.

As demonstrated by Artistic Resizing [7], we think that technical support has a great influence on the experience of designers engaged in the activity. A recent survey analyzes how designers design and program interactive behaviors with current tools [19]. Among the findings, the designers expresses that "the behavior they wanted were quite complex and diverse [...] and therefore requires full programming capabilities"; that "the design of interactive behaviors emerge through the process of exploration [...] and that today's tool make it difficult to iterate on behavior or revert to old versions"; "Details are important, and you never have them all until full implementation"; "I can represent very exactly the desired appearance. However, I can only approximate the backend behaviors"; and they want to do "Complex transitions / animations."

Based on these concerns, we propose a set of requirements for our tools. Similarly to paper prototypes in PD, tools should *maximize expression, exploration, and communication* between designers. Maximizing expression requires rich graphics, hence a toolset should be able to handle *heavy graphical* scenes, with lots of *subtle* graphical properties. Designing such scenes requires efficient design tools, such as vector graphics editors. However, in order to be usable in interactive system, the toolset should *deliver enough performance*. Maximizing exploration implies a system in which changing things (e.g. a graphical property) should be as inexpensive as possible, i.e. with as *little manipulation as possible* required to reflect the change in the subsystems.

## TOOLKIT DESIGN AND CONCEPTUAL MODEL

Designing a system that addresses all the requirements above is beyond the scope of this paper. In this work, we describe Hayaku, a tool set that partly addresses these requirements. In particular, we address *richness of expression*, *exploration*, *performances*, and *reusability*. Hayaku mainly focuses on the rendering part of the application. It also provides hooks to implement interaction with the user and communication with the rest of the application. The functional part of the application (what happens in the system when the button is pressed for instance) is out of the scope of the paper.

### General Idea: the Interaction Designer is in Charge

As said before, this activity requires designers with skills in graphic design, animation, interaction design and programming. Omniscient individuals that possess all skills are rare, if existing. In order to tackle this problem, teams include specialists in each domain, and design is distributed among the members of the team. In particular, graphical designers and computer scientists (or more precisely interaction programmer) are among the kind of specialists involved in the design of interactive software.

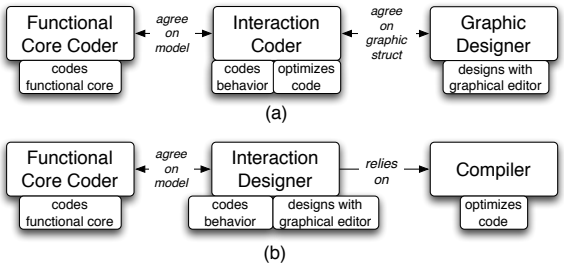



**Figure 2. Role repartition with Intuikit and XAML (a) and role repartition with Hayaku (b).**

The general idea of our approach is wider than the Intuikit approach [6]: instead of acknowledging the irreconcilability between graphical designers and interaction programmer, and maximizing communication between two different specialists, we tried to make the programmer's concerns accessible to the graphical designer. More precisely, what we target is a graphic designer that has basic programming skills, and that the tool empowers. As depicted in Figure 2, Hayaku provides the required graphical expressive power, while offloading optimizations to the graphical compiler. This turns the *interaction coder* and the *graphic designer* into an *interaction designer*. Again, we assume that the artefacts produced at this stage in the design should be done with all concerns (graphics and code) in mind, and thus by a unique person, or a very close team that share tools and artefacts. The approach is similar to Artistic Resizing: instead of describing with code the behavior of graphical elements under size change, Artistic Resizing enables graphic designers to express the behavior with means closer to their knowledge.

We provide the interaction designer with a tool chain that uses a standard vector graphics editor (Inkscape or Adobe Illustrator) as its first link. This has two advantages: the designer leverages on her experience with such tools, and she

can express graphics using the full expressive power of the tools. The other links of the toolchain consist in a compile chain that takes two inputs: graphics elements edited with the graphical editor, and abstractions of graphical element to control them. The remaining of this section enumerates the main features of Hayaku. One of the contributions of this work is the identification of those features. The goal of the paper is to present the concepts used by the tool, and show why they are adapted to the activity we target. Though there is not enough information to fully describe the system because of limited space, the concepts presented here can be used by readers if they want to design a similar system.

Abstract and Control Graphical Elements The graphic editor stores the drawings in an SVG description. SVG drawings are like "classes" of graphical objects. In order to use SVG drawings in a real application, the designer has to provide three descriptions, all written in JSON[4]. The first one is the "conceptual language" shared with the functional core coders, and serves as a bridge between the functional core and interactive graphics. As in [6], the designer and the rest of the team must agree on a common data structure, or "models" which also acts as classes of concepts. This language is illustrated by the right part of Figure 3. The second one describes how the models defined in the first description is related to SVG graphics, by connecting fields of the models to nodes and attributes in SVG drawings with a mini, data-flow-like functional language. It is similar to a stylesheet. In Figure 3, the connections are represented by the lines between the SVG part and the abstract model part. Finally, the last description is the "scene", i.e. a list of instances of the classes (not represented on Figure 3).

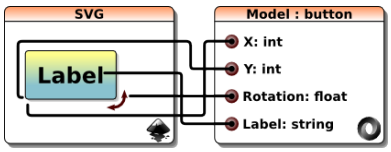



**Figure 3. Representation of the connections (the black lines) between the graphical classes (the SVG) and the model.**

Though this conceptual model of application design seems complex, it is no more than existing ways of writing code: the first JSON description can be considered as a class definition, the second one as a stylesheet, while the last one corresponds to the instantiation phase of classes at the launch of a program. The only addition is the SVG description, which corresponds to "graphical classes" definitions.

### Fast Application Generation

Hayaku includes a compiler that takes the SVG description and the three JSON files as input, and generates an application. The compiler uses various strategies to maximize compile speed and launch speed of the generated application. This allow for rapid fixes and tests, and thus efficient exploration of design.

[4]Javascript Object Notation

### Fast and Portable Code Generation

As many compilers, the graphical compiler is able to optimize the generated code. Thanks to a data-flow analysis, and user-provided hooks, the code allows the use of complex graphics (expressive power) with a rendering speed compatible with interaction. Furthermore, the compiler is able to target different graphical back-ends, such as OpenGL or Cairo. This guarantees that the design is portable.

Generate Whole Application or Embeddable Code The compiler can generate either a stand-alone application, or embeddable code. With traditional toolkits, embedding is often limited to a window that the host application displays next to its own windows. The kind of embedding that we target is more useful: graphics should appear inside an existing scene of the host application. Such embeddable code allows for creation of dynamic applications, in which the number of graphical elements is not known at compile-time. This also allows designers to use the compiler as a translation tool between SVG and a run-time environment. More generally, this transforms our toolset in a toolkit for graphical toolkit design (a toolkit of toolkits).

## USE CASES

In order to illustrate our approach, we describe how to use Hayaku to implement three different kinds of applications. Though the descriptions look like a tutorial, they enable to understand and assess how a designer is supposed to use the features provided by the tool, and help evaluate how efficient the features are at supporting the designer's activity. The first one is a basic multi-touch application that enables multiple users to move and resize simple graphical objects. It is not very rich in terms of graphics, but since it is simple, it allows for a gentle introduction and short code examples. The second one is a more graphically complex application: a resizeable keyboard with a fish-eye effect that is activated only if the size of the keyboard is too small. The last example is a generic pie-menu that can be reused in an existing application.

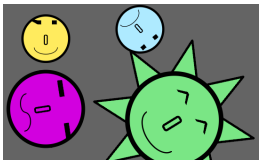


**Figure 4. A simple multi-touch application.**

### Writing a Simple Application

This test-case consists in writing a simple multi-touch application (Figure 4). The interaction consists in controlling in a simple and natural way each of the "heads" that appears on Figure 4. The properties that users of the application can control are the position, size and rotation of each shape.

For the designer, the first phase consists in defining four *graphical "classes"* (here the "head"-shapes) with Inkscape, and save them in a SVG file.

```
{  "model": "SMILEYS",
    "classes": [  {
        "name":"Object",
        "extends":null,
        "attributes": {
            "ID":"key",
            "X0":"vint", "Y0":"vint",
            "SCALE":"vfloat",
            "ROTATION":"vfloat",
            "PRIORITY":"vfloat",
            "Picked_Key":"vint" }},
     { "name":"Object_0",
        "extends":"Object",
        "attributes": {}}]}
```

**Figure 5. Model of the multi-touch widget.**

```
{"model":"SMILEYS",
  "objects": [
  {"className":"Object_0",
    "file":"demo.svg",
    "graphicalItems": [
    {"name":"smiley_svg",
      "connections":
      {"X0":"smiley_svg.transform.tx",
        "Y0":"smiley_svg.transform.ty",
        "SCALE":"smiley_svg.transform.scale",
        "ROTATION":"smiley_svg.transform.rotation",
        "PRIORITY":"smiley_svg.transform.priority"},
      "picking":
      {"Picked_Key":"smiley_svg"}}]}]}
```

**Figure 6. Connection between the model of the multi-touch widget and the graphic parts (smiley_svg).**

```
{  "name":"Smileys",
    "model":"SMILEYS",
    "content": [
    {  "type":"Object_0",
        "attributes": {
            "ID":0,
            "ParentID":0,
            "X0":100, "Y0":100,
            "SCALE":0.5,
            "ROTATION":0.0,
            "Picked_Key":-1 }}] }
```

**Figure 7. Instantiation of the multi-touch widget.**

```
def translate(self,dx,dy):
    self.x0.set(self.x0.eval() + dx)
    self.y0.set(self.y0.eval() + dy)
def rotate(self, dr):
    self.rotation.set(self.rotation.eval() + dr)
def zoom(self, z):
    if self.scale.eval() + z >= 0.1:
        self.scale.set(self.scale.eval() + z)
```

**Figure 8. The Python code of the three commands to control the graphical objects.**

The third phase consists in defining the *connections* between the model and the graphical part (Figure 6), again in a JSON file. Connections are straightforward and need no explanation. The fourth description pertains to the *scene*, in another JSON file. This file consists in instantiating the different elements of the graphical scene (Figure 7).

The designer has to provide the reactive part of the application, i.e. the connection between input events and reaction of the graphical objects. Since Hayaku focuses on the rendering part only, it does not provide any multi-touch capabilities. Rather, it is up to the designer to describe with the run-time language and input toolkit how events act on the conceptual model, by updating the corresponding fields of the instances. However, when generating the code corresponding to the conceptual model, the toolset offers the possibility to concatenate user-defined code. This enables the designer to abstract behavior (see Figure 8). Furthemore, Hayaku provides a picking mechanism that can be called from user-defined code.

In order to test and launch the application, the interaction designer edits a Python script that contains a call to the function *load* with the three JSON files as arguments (the model, the model-to-svg connection, and the scene). She then launches the command *hayaku* with the script as a parameter. If the compile phase succeeds, Hayaku launches the generated application.

The compilation time for this example is 2.2 seconds the first time. Further recompilations requires 1.9 secs only. The first time of compilation is longer due to some tools that need to be embedded in the final application and that does not need to be recompiled each time a change occurs (OpenGL shaders and utility functions). The application takes less than one second to launch, and runs at 515 frames per second (see Table 2). Again, this application is simple and not demanding in terms of computation power. Still, it shows that the toolkit is reactive enough to deal with high-rate incoming data.

### A Fish-eye Keyboard

The second application is a 40 auto-expanding keys keyboard, designed for motor-disabled users (Figure 9) [22]. The keyboard consists in two parts: the keyboard itself, and a one line screen to display the result. The caps-lock key is fully functional: the key mapping changes accordingly. The key *"123"* toggles the numeric mode. Finally, the keyboard can be resized, and at low sizes the keys close to the cursor expand thanks to a fish-eye effect [10].


**Figure 9. The test application in action.**

This example demonstrates the ability of the toolkit to handle rich graphics with high rendering performance. The design of the keyboard uses a full vectorial description for its components. This leads to high quality graphics even when the keyboard is resized. The design also uses rich graphic properties: gradients, transparency, shadows...

#### *Realisation*

The graphical part of the application has been realised with Inkscape (Figure 10). In a first SVG file, the designer creates a key by using eight separate graphical layers. The layers are grouped and named in a unique SVG component. In order

to build the global composition, the keys are then cloned, organized and modified to generate an artwork of the final keyboard. The creation of the upper area, including the text display, the backspace key and a background with a gradient completes this artwork. The whole keyboard contains 400 graphical elements.

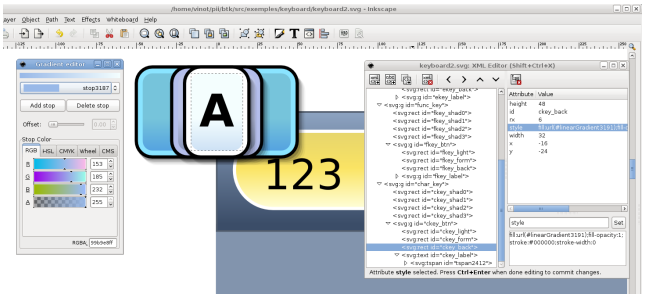



**Figure 10. The SVG description of the different components of the keyboard, realized with Inkscape.**

Once the global composition is satisfactory, three examples of the different type of keys are put in a separate SVG file, to serve as "graphical classes" : *char_key*, *func_key* and *enter_key*. The graphical components correspond to the component described in the model, and are named accordingly. The blocks that describe the background and the display of the result are also added to this file. A parent class *Key* has been defined to handle the common properties of the different keys. The class is inherited by the different types of key (*char*, *func* and *enter*).

The layout of the keyboard is given in the JSON *scene* file. However, Hayaku does not provide a visual editor for the scene. Thus, the designer has to provide it. Since the production of this file can be laborious, a script has been written to produce it. This script allows the interaction designer to rapidly change the layout of the keyboard by changing some variables in the script, instead of a bunch of values and parenthesis into the JSON file.

The fish-eye effect is implemented by computing the distance between the cursor and each key, and by using this distance to set the scaling property of the key accordingly. Each time the cursor moves, a redraw is triggered, and the key is scaled with is current scale before being drawn.

### A Generic Pie-menu

To assess that Hayaku can be considered as a toolkit of toolkits[5], we implemented a generic pie-menu (Figure 11). The objective was to provide an implementation-independent description in order to use it inside an actual, existing application (Figure 12). The pie-menu we designed includes a feedback when flying over a slice: the underlying slice is enlarged. Thus we can not use a mere circle, but several distinct slices. We also need to be able to control the number of elements inside the menu.

[5]here, Hayaku can be considered as a toolkit for building a widget

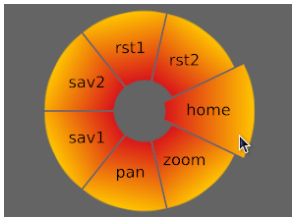



**Figure 11. The pie-menu in action.**

#### *Realisation*

The design in itself resembles the design of the keyboard: we designed the pie-menu to be a set of slices. Each slice has 7 main graphical parameters: a *position*, a *label*, an *angle*, an *internal radius*, an *external radius*, a *rotation*, and a *color* parameter. To describe the scene, we wrote a script similar to the one that generated the keys in the keyboard. The script generates the slices and their parameters according to the number of slices.

The behaviour part maps the picking value of each slice with a callback that changes the internal radius, the external radius and the color as needed. High-level events, such as "*menu 7 has been selected*", have to be generated by the behaviour part, since Hayaku only provides the graphical part of the application.

#### *Embedding in an Existing Application*

We have embedded the pie-menu into an existing radar-like application for Air Traffic Control (see Figure 12). This application is written in C++ and makes extensive use of OpenGL. The application is extensible, and provides a mechanism for loading dynamic external libraries. We used this mechanism to plug our pie-menu into this system.

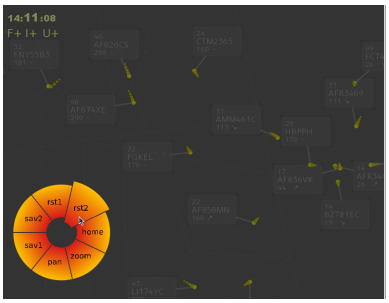



**Figure 12. The pie-menu inside a real application.**

The steps involved were the following. First, we had to write a C++ class that interacts with the dynamically loaded objects generated by Hayaku. This class is the glue that links the host application and the generated interactive graphics, and factorizes the setup code for all embedded Hayaku code. Then we wrote a subclass specific to the pie-menu, to handle the pie-menu behaviour with respect to user interaction. This subclass represents 112 lines of code. It is a transcription in C++ of previously written Python code, developed during the prototyping phase of the pie-menu widget. As

Figure 12 shows, the pie-menu smoothly integrates into the host application, and does not reduce the frame rate.

This use case shows that it was possible to externalize the creation of widgets and reuse them in other applications. However, in general, existing systems do not support extensions with external dynamic plug-in: in this case, the code generated by Hayaku must be embedded at source-level. The glue between the original code and the graphical part is simpler (just a "#include" at the beginning). Drawing is initiated by calling the exported *draw* function.

## TOOLKIT IMPLEMENTATION

### How the Toolset Works

The command *hayaku* automatically calls the GrC. The GrC then creates a directory named *BUILD* in which it places all its productions. The JSON files are transformed into Python ones, and a set of C files and their headers are written. Then, the GrC calls *gcc* to compile those C files and produce the object files that can be embedded into C applications. It generates a dynamic library that can be either linked to the runtime of the GrC, or embedded into an existing application.

To reduce compile time, the compiler is able to detect parts that have been modified between two successive compilations, and compiles those parts only. In addition, we designed a monitoring system on the files, and the recompilation occurs automatically whenever a file is modified and saved. The change is automatically reflected in the generated application while it is still running. For example, changing the color of one of the shapes in the example above with Inkscape, and saving the SVG file automatically updates all shapes of this class in the running application. This illustrates the advantage of separating graphics from behavior and using data-flow mechanisms: since the graphical pipeline is clearly delimited, the toolset is able to trigger it at any time, without affecting the behavior of the whole application. Such tools reduce the time needed between envisioning an idea and testing it.

### Generation of Portable Code

As we already said in the previous sections, the designer produces the graphical shapes thanks to SVG files. The abstractions and connections between those graphical shapes and the models are given through JSON files. Then, Hayaku loads them into the GrC.

The GrC in itself is written in Python. The GrC is able to produce different types of outputs, in terms of target language and run-time (currently C and Java), and in term of graphical backend (currently OpenGL, and partly Cairo). To be able to reuse the code of the transformations, we implemented our own partial class mechanism. We separate the description of the intermediate languages and the transformation between them. At the beginning of the compile chain, the GrC chooses which languages and transformations it needs to produce the final code by attaching the transformation functions to the descriptions nodes. The trees that are generated can then be transformed just by visiting each node. This mechanism allows us to modularize the graphical compiler and thus to plug different behaviour at different stages as needed.

### Generation of Static and Semi-static Code

Most examples are instances of application in which the number of objects is not variable (sliders, pie-menus, keyboard). For other types of applications, such as radar image where the number of flights is in theory not bounded, the data-flow architecture does not allow for simple description and handling of dynamic creation of objects. In this case, Hayaku provides two strategies.

The first one is to consider the number of elements to be displayed bound by an upper limit [24]. This requires to start the application with a pool of available invisible graphical objects, which are allocated to any new data that appear during run-time. In practice, this strategy works well: for example, the number of flights in a sector is bounded by regulation agencies in order to enable a limited numbers of controllers to handle the traffic. It comes at the expense of internal handling of invisible objects (which may hinder performance uselessly) and longer compile time. But the benefits outweigh the drawbacks, since it helps keeping the application simple to write and understand.

The second strategy consists in generating pieces of specific interactive graphical code that can be reused in a larger program. In the radar image, this would consist in designing the graphics for a single flight, and generating the corresponding display code. The main program would then manage creation of new flights and deletion of disappearing ones, and use the display code whenever necessary. With this solution, the compile time is reduced, since the graphical code is not unrolled as in loop unrolling for instance, and the constraint of the upper limit of objects is removed.

Generated code must follow a number of requirements to make it embeddable. First, the generated code has to keep the state of the application. For instance, when working with OpenGL applications, the drawing code has to keep the pipeline in the same state it was before its use. A second requirement is to produce "human readable" code. Since most of the time a designer will connect the generated code to the other application, the names of the functions that are exported have to be understandable by the programmer. For instance, *set0_25_2_1* is less readable than *set_component0_key25_backgroundColor_red*.

### Picking Support

The generated code must provide a way to send back information. For instance, when the end-user moves the mouse, the code has to inform the caller that the picking state changed. Hayaku provides a picking mechanism, together with a callback system. The host application has to register callbacks if it wants to be notified by the graphics code, or by the underlying dataflow. Care must be taken when handling picking. For example, a usual picking algorithm consists in rendering the scene in a tiny rectangle around the cursor, and storing each graphical object that owns pixels actually rendered in the rectangle. Applying the same algorithm

in a multitouch application requires as many passes as the number of touches, which is costly. Instead, we used a one-pass color-keying algorithm [12]. Each graphical shape is assigned a unique color in an associative array, and rendered with their unique solid color in an off-screen buffer. Picking shapes consists in reading back the color of the pixel under each touch, and retrieving the corresponding shape from the color with the associative array.

## PRELIMINARY EVALUATION

As with any method that aims at supporting design, evaluating a toolset requires controlled experiments, with multiple design teams under different conditions (with or without the tested toolset for example). Such an experimentation is a heavy task, and is beyond the scope of this paper. However, we provide in this section a preliminary evaluation in terms of descriptive power, performance, and usability.

### Descriptive Power

We provide two dimensions of analysis to evaluate the descriptive power of the toolkit: the size of the class of visualizations that can be described by the toolkit in a reasonable amount of work, and the simplicity of the description of typical applications. A toolset must target the right balance between the class size and simplicity. A thin class may indicate that the toolset is so specialized that the benefits provided are not very significant. On the other hand, expanding the class usually comes at the expense of simplicity.

*Class of Application:* previous work showed that the GrC is able to handle basic WIMP interaction (sliders) and graphical scene with a large number of objects, such as a radar image. We showed with the use-cases of this paper that Hayaku can implement multiple types of interactive graphical software: interactors (pie-menus), graphically rich interactive software (fish-eye keyboard), and multitouch applications.

As said before, most examples are instances of application in which the number of objects is not variable (sliders, pie-menus, keyboard). For other types of applications, such as radar image where the number of flights is in practice bounded, a strategy consists in picking objects in a pool of available invisble objects. Hayaku enables to use a second strategy that relies on embeddable, generated code, thus expanding the class of applications.

Using a graphical editor also enables the designer to expand the class of representation he can employ. However, we did not try to design very dynamic applications such as graphical editors with Hayaku because we think that Hayaku is not made for that kind of applications. We suspect that writing such systems would require to twist the conceptual model of application design so much, that it would be too cumbersome to do.

*Simplicity*: Despite our research in the literature, we could not find a clear definition for simplicity. Thus, we measured it in terms of compactness of the code required to describe interactive graphics, by providing the number of lines of code (LOC) of previously described examples. (Table 1). As said before, the JSON description of the scene (the graphical components of the interface) has been judged as "laborious", and a Python script to produce it has been required. It corresponds to the "generator" column. For example, the 890 LOC for the keyboard have actually been generated by the 210 lines of code generator. As we can see, the amount of code is in the hundreds, which is low considering the richness and variability of the three examples.

| use case | conceptual model | model to SVG | scene | generator of the scene |
|---|---|---|---|---|
| multi-touch | 43 LOC | 90 LOC | 54 LOC | $\emptyset$ |
| keyboard | 129 LOC | 199 LOC | 890 LOC | 210 LOC |
| pie-menu | 40 LOC | 42 LOC | 102 LOC | 46 LOC |

**Table 1. The number of lines of code (LOC) of the different examples.**

### Performance

In Table 2, we show the performances of the three use-cases, compiled with Hayaku, and rendered through OpenGL. For each example, we show the frame rate of the produced code (C+OpenGL), and the time needed to compile it. We differentiate "first compile-time" from "re-compile time", because Hayaku caches some computation between two consecutive compile phases (text fonts for example). The most significant time is the re-compile time, since a designer using Hayaku will spend most of her time doing small increments to her description, and will launch recompilation from time to time.

| use case | frames per second | first compile time | re-compile time |
|---|---|---|---|
| multi-touch | $\sim$515 f.p.s. | 2.2 sec | 1.9 sec |
| keyboard | $\sim$136 f.p.s. | 29.1 sec | 8.6 sec |
| pie-menu | $\sim$400 f.p.s. | 10.2 sec | 2.9 sec |

**Table 2. The performances of the different examples.**

If performances may not be as good as expected, they could be much higher (8.6 sec re-compile time for the keyboard). The implementation of the toolkit we show here is a prototype (written in the Python language), and could be improved in many ways. For instance, the produced OpenGL code does not use Vertex Buffer Objects, which could significantly improve the run-time performances. In addition, the internal data structures of Hayaku and the GrC (graphs of tiny Python objects) should be changed to decrease compile time.

### Usability

Evaluating the usability of a toolkit is an open research problem [20]. For this purpose, we discuss how Hayaku ranks against Cognitive Dimensions of Notation [11], which help make explicit what a notation (i.e a language) is supposed to improve, or fails to support. Cognitive dimensions are based on activities typical of the use of interactive systems. We chose to evaluate the following activities: *incrementation*, *transcription*, *modification*, and *exploratory design*; along the following dimensions: *closeness of mapping, hidden dependencies, premature commitment, progressive evaluation, abstraction, viscosity*, and *visibility*.

*Closeness of Mapping:* the designer creates (*incrementation*) drawings directly into a graphical editor: it is very *close* to the final product, at least closer than textual graphical language. This allows the use of existing *exploratory design* tools (inkscape), and thus maximizes this property. *Modification* of the graphics is eased since it modifies in turn an SVG file that keeps the same properties (e.g. naming), which in turn is compiled i.e. transformed computationally. Porting can be considered as a *transcription*, and is efficient thanks to the use of a compiler with multiple front-ends and back-ends. The front end of the compiler is the conceptual model JSON file. Since the interaction designer designs the conceptual model, she can make it as close as possible to the domain she models. Hence, closeness of mapping is maximized. However, setting the link between the graphics, the conceptual models, and the data-flow language requires a switch of notation (a graphical editor vs a textual notation).

*Hidden Dependencies:* the dataflow we provide is not entirely visible. It is difficult for the designer to know exactly what happens once the models and transformations are given. However, the designer is mostly interested in the part of the data-flow he wrote. The part of the data-flow generated by the compiler is less susceptible to be read and understood, except for debugging purpose. *Premature Commitment*: using a graphical compiler inherently prevents premature commitment. For example, changing the run-time environment can occur at any time during the design process. Furthermore, changing a property of the graphics may require a simple recompile to be reflected in the application. Moreover, as Hayaku relies on style-sheets to link the graphical model to the graphical shapes, the design can be rewritten several times without having to rewrite the behaviour part. However, the structure of the graphics must not change too often, since other descriptions rely on it (see viscosity). *Progressive Evaluation:* evaluating a recently modified graphics is immediate. However, evaluating the behavior with respect to the interaction requires to launch the software. Clearly, a tool such as artistic resizing is needed for this kind of activity and concerns.

*Abstraction:* Hayaku relies on JSON files to *abstract* the graphical model, the connections and the graphical scene. However, if this language is well adapted to represent abstract data, and forces the user to keep it abstract, it is not very well adapted to the human that needs to write it into his/her text editor. In particular, the connection between the models and the graphics would be better defined directly in a graphical editor.

*Viscosity:* the conceptual model requires all graphical elements to be declared in the JSON file. Hence, if a graphical element is used multiple times (such as the key element in the keyboard example), a change in the "prototype" requires propagating the change in all instances of that element. A solution to this *viscosity* problem is to design a small program that generates all instances from a prototype in a JSON file. This program can be considered as another link in the compile chain, and helps abstract concepts from the conceptual model of the application to be designed.

A change in the conceptual model itself must be reflected into the connection description, and the scene description. This is the problem that the programmer of a C++ class encounters when he adds a field for example: he has to update all calls of the class constructor if a parameter to set up the field is required. Various mechanisms exist to cope with this problem (e.g. a default value), but none is implemented in Hayaku. Similarly, a change in the graphical structure (i.e. the hierarchy of SVG elements) can have a large impact on the model-graphics connection description.

*Visibility*: currently, the *visibility* of the toolkit is limited. For example, JSON files tend to be verbose and long, which hinders searching or exploratory understanding.

## EARLY FEEDBACK FROM DESIGNERS

We provided Hayaku to the graphical designer of the original Fish-Eye Keyboard, and we asked him to recreate it. This designer is used to both design graphics and write interaction code. The designer praised the reliability of the rendered scene. Since Hayaku relies on a graphical compiler, the final generated code does not suffer from a trade-off between speed and power of expression. The final rendered scene is then very close to a static one, produced by Inkscape for instance. Thanks to the expression power of SVG, the graphic designer is not limited when dealing with graphics.

The designer found that one of the most interesting thing was to design the graphical objects by keeping in mind their graphical behaviour during the interaction. This behaviour has been defined by targeting the graphical properties that need to be *connected* to the models in the graphical scene. The evolution of the parameters are then described, either "relatively" with a mathematical expression (similar to the one-way constraints in Garnet [25]), or with a value computed by the behaviour part. For example, the anchor of the shape of a key depends on its width ("FORM_X0": "(self. WIDTH - 100) / -2"): since the width depends on the distance with the cursor, the anchor is updated automatically. Considering all the inputs and outputs of the generated application as a data flow simplified the work of the designer. For instance, implementing the global resize of the keyboard took around 10 minutes, the time needed to understand and implement the solution to connect the two variables screen_width and screen_height to the application. The "connections" allow the graphic designer to quickly build complex behaviour, such as the "fish-eye" function of the keys.

However, there still are some pitfalls. The main problem was the hand writing of the different JSON files. This has been judged as laborious, since the coherence between those files had to be maintained manually. Furthermore, writing a JSON file for the scene is also annoying, since a scene can contain many similar elements. As explained, a solution to this problem is to write a script that generates the scene, which makes it more controllable.

## CONCLUSION

In this paper, we have identified that there is a lack of tools to support designers in producing graphically rich, finely tuned

and highly reactive graphical applications. We have presented Hayaku, a toolset that aims at supporting this activity, by turning the interaction coder and graphical designer into an interaction designer. The interaction designer writes the program in a high-level, known language (SVG) and through JSON files that abstract the graphical elements. He then compiles it into an runnable application or embeddable code.

Like the keyboard example shows, the compile time hinders design exploration, and must be improved. We have developed Hayaku in Python in order to prototype it rapidly, and we are aware that parts of the code are sub-optimal (notably trees traversal). Many optimizations can be done to improve that part of the toolset. Future works also include expanding the sets of back ends, both for graphics platform and languages. Finally, using multiple JSON files as a description language is cumbersome, especially when describing the connection between models and graphic models. Specialized tools must be designed, such as a graphical editor.

## REFERENCES


1. Appert, C., and Beaudouin-Lafon, M. SwingStates: adding state machines to Java and the Swing toolkit. *Software: Practice & Exp. 38*, 11 (2008), 1149–1182.
2. Beaudouin-Lafon, M., and Mackay, W. Prototyping tools and techniques. In *The Hum. Comp. Inter. Handbook*, A. Sears and J. A. Jacko, Eds. CRC Press, 2007.
3. Bederson, B., Grosjean, J., and Meyer, J. Toolkit design for interactive structured graphics. *IEEE Transactions on Software Engineering 30*, 8 (aug. 2004), 535 – 546.
4. Bederson, B. B., Meyer, J., and Good, L. Jazz: an extensible zoomable user interface graphics toolkit in Java. In *Proc. of UIST '00* (2000), ACM, 171–180.
5. Buxton, B. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufman, 2007.
6. Chatty, S., Sire, S., Vinot, J.-L., Lecoanet, P., Lemort, A., and Mertz, C. Revisiting visual interface programming: creating GUI tools for designers and programmers. In *Proc. of UIST '04* (2004), ACM, 267–276.
7. Dragicevic, P., Chatty, S., Thevenin, D., and Vinot, J.-L. Artistic resizing: a technique for rich scale-sensitive vector graphics. In *Proc. of UIST '05* (2005), ACM, 201–210.
8. Dragicevic, P., and Fekete, J.-D. Support for input adaptability in the ICON toolkit. In *Proc. of ICMI '04* (2004), ACM, 212–219.
9. Fekete, J.-D. The InfoVis Toolkit. In *Proc. of InfoVis'04* (October 2004), IEEE Press, 167–174.
10. Furnas, G. W. Generalized fisheye views. *SIGCHI Bull. 17*, 4 (1986), 16–23.
11. Green, T. R. G. Cognitive dimensions of notations. In *Proc. of HCI'89* (1989), Cambridge University Press, 443–460.
12. Hanrahan, P., and Haeberli, P. Direct WYSIWYG painting and texturing on 3D shapes. In *Proc. of SIGGRAPH'90* (1990), ACM, 215–223.
13. Hartmann, B., Yu, L., Allison, A., Yang, Y., and Klemmer, S. R. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In *Proc. of UIST '08* (2008), ACM, 91–100.
14. Heer, J., Card, S. K., and Landay, J. A. Prefuse: a toolkit for interactive information visualization. In *Proc. of CHI '05* (2005), ACM, 421–430.
15. Lattner, C., and Adve, V. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proc. of CGO'04* (Mar 2004).
16. Mertz, C., Chatty, S., and Vinot, J.-L. The influence of design techniques on user interfaces: the DigiStrips experiment for air traffic control. In *Proc. of HCI Aero IFIP 13.5* (2000).
17. Muller, M. Participatory design: The third space in HCI. In *The Hum. Comp. Inter. Handbook*, A. Sears and J. A. Jacko, Eds. CRC Press, 2007, 1061–1081.
18. Myers, B. Challenges of HCI design and implementation. *Interactions 1*, 1 (1994), 73–83.
19. Myers, B., Park, S. Y., Nakano, Y., Mueller, G., and Ko, A. How designers design and program interactive behaviors. In *Proc. of IEEE VL/HCC '08* (2008).
20. Myers, B. A. Usability issues in programming languages. Tech. rep., School of Computer Science, Carnegie Mellon University, 2000.
21. Myers, B. A., and Rosson, M. B. Survey on user interface programming. In *Proc. of CHI* (New York, 1992), CHI '92, ACM, 195–202.
22. Raynal, M., Vinot, J.-L., and Truillet, P. Fisheye keyboard: Whole keyboard displayed on small device. In *Proc. of UIST '07: poster session* (Oct 2007).
23. Snyder, C. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces (The Morgan Kaufmann Series in Interactive Technologies)*. Morgan Kaufmann, April 2003.
24. Tissoires, B., and Conversy, S. Graphic Rendering Considered as a Compilation Chain. In *DSV-IS'08* (2008), no. 5136 in LNCS, Springer, 267–280.
25. Vander Zanden, B. T., Halterman, R., Myers, B. A., McDaniel, R., Miller, R., Szekely, P., Giuse, D. A., and Kosbie, D. Lessons learned about one-way, dataflow constraints in the Garnet and Amulet graphical toolkits. *ACM Trans. Prog. Lang. Syst. 23*, 6 (2001), 776–796.

# FromDaDy: Spreading Aircraft Trajectories Across Views to Support Iterative Queries

Christophe Hurter, Benjamin Tissoires, and Stéphane Conversy

**Abstract**—When displaying thousands of aircraft trajectories on a screen, the visualization is spoiled by a tangle of trails. The visual analysis is therefore difficult, especially if a specific class of trajectories in an erroneous dataset has to be studied. We designed FromDaDy, a trajectory visualization tool that tackles the difficulties of exploring the visualization of multiple trails. This multidimensional data exploration is based on scatterplots, brushing, pick and drop, juxtaposed views and rapid visual design. Users can organize the workspace composed of multiple juxtaposed views. They can define the visual configuration of the views by connecting data dimensions from the dataset to Bertin's visual variables. They can then brush trajectories, and with a pick and drop operation they can spread the brushed information across views. They can then repeat these interactions, until they extract a set of relevant data, thus formulating complex queries. Through two real-world scenarios, we show how FromDaDy supports iterative queries and the extraction of trajectories in a dataset that contains up to 5 million data.

**Index Terms**—visualization, iterative exploration, direct manipulation, trajectories.

## 1 Introduction

In the Air Traffic Control (ATC) field, analyzing traffic or devising new ways of managing airspace requires trajectories analysis. An aircraft trajectory is a record of positions of an aircraft in a given airspace (3D+time plus other information such as identifier, speed etc). As such, trajectories are multidimensional data. Air Traffic stake-holders regularly analyze traffic to:

- understand past conflicts and then improve safety with adequate evolutions,
- assess new onboard and ground safety systems and the resulting aircraft trails,
- devise new air space organization and procedures to handle traffic increase,
- compare trails with environmental considerations (fuel consumption, noise pollution, vertical profile comparison),
- study profitability from a business trajectory point of view (number of aircraft on a specific Flight Route per day, number of aircraft that actually landed at a specific airport…),
- filter and extract trajectories in order to re-use them (this task will be later illustrated in this paper in the section on trajectory extraction for Air Traffic Controllers' training).

Formulating queries over trajectories in a declarative, textual-language based manner, such as a SQL, is hard. Even if it is possible to select trajectories that flow over specific locations, it is very difficult to specify features like "select trajectories where this part of the trajectory is straight" or "where this part has a constant climbing rate"… Thus, visual analysis remains the only way to detect relevant trajectory features efficiently.

- *Christophe Hurter is with DSNA/DTI R&D, ENAC and IRIT/IHCS E-Mail: christophe.hurter@aviation-civile.gouv.fr.*
- *Benjamin Tissoires is with DSNA/DTI R&D, ENAC and IRIT/IHCS E-Mail: tissoire@cena.fr.*
- *Stéphane Conversy is with ENAC and IRIT/IHCS E-Mail: stephane.conversy@enac.fr.*

*Manuscript received 31 March 2009; accepted 27 July 2009; posted online 11 October 2009; mailed on 5 October 2009.*
*For information on obtaining reprints of this article, please send email to: tvcg@computer.org .*

Trajectories are numerous and tangle: one-day's traffic over France for example, represents some 20000 trajectories ( Fig. 1). When dealing with trajectories, users must perform dynamic requests (response time < 100 ms [15]) on a huge multi-dimensional dataset (>1 million data). In addition to the data size problem, users have to deal with a dataset that contains many errors and uncertainties: recording is done in a periodic manner (in our database: a radar plot per aircraft every 4 minutes), but a plot can be missed, or have erroneous values because of physical problems that occurred at the time of recording. The problem we address in this paper is to find a way to express these queries, simply and accurately, given the constraints of size and uncertainty of the datasets.

We have developed FromDaDy (which stands for "FROM DAta to DisplaY"), a visualization tool that tackles the challenge of representing, and interacting with, numerous trajectories involving uncertainties. FromDaDy employs a simple paradigm to explore multidimensional data based on scatterplots, brushing, pick and drop, juxtaposed views and rapid visual configuration. The fundamental new aspect of FromDaDy compared to existing visualization systems, is to enable users to *spread* data *across* views. Together with a finely tuned mix between design customization and simple interaction, users can filter, remove and add trajectories in an iterated manner until they extract a set of relevant data, thus formulating complex queries.
The remainder of this paper is organized as follows. First, we present relevant related work. Then we list the design requirements to fulfill the trajectory analysis task. Next, we describe FromDaDy features and justify our implementation choices. Finally, we outline the strengths of FromDaDy with two specific data extraction scenarios.

## 2 Related work

FromdDaDy proposes a simple model of interaction that, compared to existing models of interaction, provides more explicit support for incremental data exploration, visual configuration and Boolean operations. Our work is based on many previous research publications on visualization and interaction with multidimensional data (Spotfire [1], Tableau/Polaris [16], GGobi [17], TimeSearcher [11]).

### 2.1 The dataflow model

Card, Mackinlay and Shneiderman [6] proposed a model that describes visualizations as a data flow sequence from the raw data to the views. This data flow model is still widely used in a lot of visualization software (SpotFire [1], VQE [7] , InfoVis Toolkit [9], ILOG Discovery [3], nVizN [19]…).

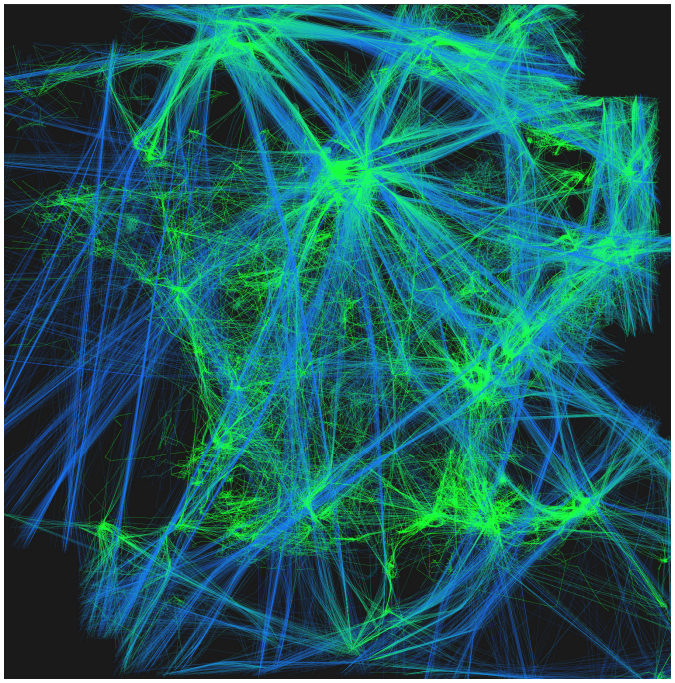


Fig. 1 : One-day's record of traffic over France. The color gradient from green to blue represents the ascending altitude of aircraft (green being the lowest and blue the highest altitude). The French coastline is apparent here in terms of sightseeing by light aircraft and the straight blue lines represent high altitude Flight Routes.

### 2.2 Simple filtering and selection

Though originally designed for data exploration, Dynamic Queries [2] represents the seminal work in query design. The associated "range-slider" widget, allows for fast, incremental manipulation of ranges, with immediate effect [15] . As such, a range-slider reifies a simple query, which filters out data outside the range.

Some systems allow data to be selected by defining an area over graphical entities, which changes their appearance (for example, they are reddened). In a multiple view system, such as a scatterplot matrix, selected data appear highlighted both in the view manipulated by the user, and in the other views, making it possible to understand the relationships between selected data.

Interactions for selecting data include one-by-one designation [12], rubber-band rectangle [12][8], lassoing [8] or brushing [4]. Various systems propose enhanced brushing techniques, such as XmdvTool [18]. However, they require a complex interface to tune parameters, which hinders rapid iteration. For example, an "erase-data" mode in XmdvTool is accessible, but only through a dialog box.

### 2.3 Defining filtering and selection

All tools enable the user to define a selection, but again in various degrees. With Dynamic Queries, users can point to a range-slider previously manipulated to adjust the range. "Rolling the Dice" [8] makes it possible to "sculpt" queries, but only by defining a new selection to be combined with existing ones. Though not fully explained, it seems that redefining a selection requires defining a completely new one: it does not seem possible to resize a rubber rectangle or modify a lasso shape. XmdvTools allows the user to add a new brush over an existing one, but does not allow removal of parts of the stroke [18]. TimeSearcher allows the user to select time series with movable boxes [11].

### 2.4 Multiple filtering and selections, Boolean operations

Multiple range-sliders implicitly combine their queries into a single one, implementing a Boolean "and" operation. Some systems allow multiple selections (sometimes called "layers"), differentiated by colors. This enables the user to find patterns between the different groups of selected data. The combination of selections is done by the visualisation of a mix of differently colored shapes. Thus users visually apply a "xor" operation when seeking groups of isolated shapes, while they apply an "and" operation when they try to group differently-colored shapes.

In some systems, users can explicitly define how selections are combined by choosing a Boolean operation: the resulting selection is then highlighted with yet another color. The interaction uses either a specific tool [18], or a specific button of the interface at the start of the interaction [13]. "Rolling the Dice" [8] reifies selections into stacked rectangles that enable the user to combine selections by dragging and dropping one rectangle onto another. The choice of which Boolean operation to apply is made by dragging either with the right button (and) or the left button (or). Once executed, the two selections are merged into one, and they cannot be manipulated any further.

### 2.5 Views organization and navigation

Matrix scatterplots are scatterplots arranged in a matrix, so that every scatterplot on a row (or column) shares the same dimension on the X (or Y) axis. As each dimension is spatially matched to the others, users can detect spatial patterns at a glance. In addition, there is no need to navigate between views, as all of them are displayed at once. This enables users to switch rapidly between views, so as to interact with the view that is the most adapted to the problem at hand. By contrast, a traditional visualization system offers few ways to display multiple views, and forces the user to switch between views with standard window manipulation.

However, the size of scatterplots matrix scales quadratically with the number of dimensions, and results in thumbnail views that are difficult to visualize and interact with. Furthermore, even if interaction-free navigation requires finding a particular scatterplot in the matrix, this sometimes takes time; the user has to find the row and the column of the two dimensions to explore, and then find the scatterplot at the intersection between the row and the column. Designed to overcome this problem, "Rolling the Dice" [8] offers a number of interactions to navigate from one scatterplot to another, and displays a rolling dice-like animation when switching between views. However, "Rolling the Dice" displays only one scatterplot at a time (with geometrically transformed selections already made in other views). This makes interaction with previous selections longer, as it requires the user to look back and switch to a more appropriate view.

## 3 DESIGN REQUIREMENTS

This section presents the design requirements required to achieve trajectory exploration. The majority of our tasks consist in finding real world trajectories with a specific set of features. This contrasts with the traditional InfoVis tasks, where the goal is to discover trends or outliers. Trajectory features are difficult to specify for two reasons. First, they are often only specifiable with visual features (straight lines, or general shape). Furthermore, users often explore the queries as much as they explore the data: in the course of exploration, users discover that the set of features they thought relevant has to be adapted, either because they were false, or because

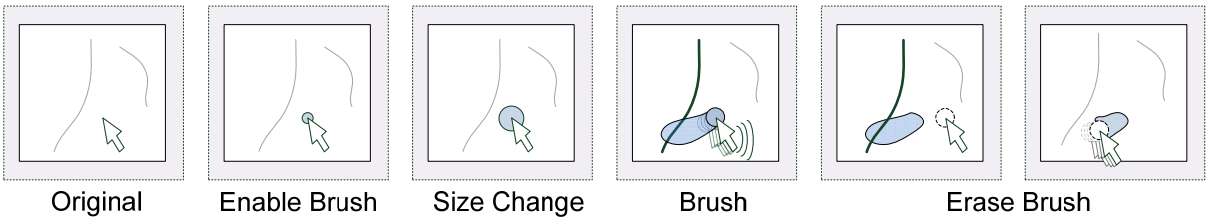



Fig. 2 : The brushing interaction allows the user to select trajectories by brushing them with a size configurable tool.

they cannot figure out how to query them efficiently. Hence, the system must permit the customization of views so as to offer multiple means of understanding and querying the data visually. It should allow for a quick change of mapping between data and visual dimensions. Often, the set of interesting trajectories for a particular task can only be described by extension: hence, the system must also support iterative selection design, i.e. the ability to store a temporal state of a selection and to be able to improve it later. Trajectory databases are huge and multidimensional (more than 500000 records with more than 10 fields: aircraft's name, speed, location…). The system should be able to handle this amount of data, and display graphical entities with a frame rate compatible with smooth interaction. As said earlier, our database contains many errors and uncertainties; thus the user must be able to figure out if the displayed trajectory is reliable or not. If not, users must understand why.

## 4 RADAR DATASET

Our radar dataset contains recording of aircraft parameters at a given time (Table 1). This dataset may contain many other fields, but we present here the most important ones. Records are linked through the aircraft identifier (provided by radar tracking). Points are gathered to form trajectories.

| Field name | details |
|---|---|
| Latitude | Latitude of the aircraft at a given time |
| Longitude | Longitude of the aircraft at a given time |
| Flight Level | Altitude of the aircraft |
| Time | The time of the record |
| Speed | The aircraft speed |
| Track ID | The unique identifier of the aircraft |

**Table 1 :** Record field names and semantics.

The trajectories dataset contains many errors:

- The radar tracking system is faulty when an aircraft has a very low altitude,
- The onboard system may emit temporally wrong information (aircraft ID and altitude) ,
- The flight route used by aircraft may not correspond to the current aircraft heading (due to metrological considerations, traffic optimization or safety reasons).

These errors are very important since they can highlight a radar loss detection area, or onboard technical problems. Errors can easily be detected visually when they create outliers or discontinuities in visualization: e.g. the aircraft altitude suddenly jumps to zero then back to high.

The dataset also contains uncertainties, which are due to the sampling rate of the aircraft position. Our available dataset contains aircraft positions every four minutes. Therefore the actual aircraft position between two consecutive positions is unknown. For exemple, aircraft having landed may stop at a high altitude (the last detected position lasting four minutes).

## 5 SYSTEM DESCRIPTION

This section details FromDaDy basic features for trajectory exploration tasks.

### 5.1 Visual configuration

FromDaDy uses the data flow model, through a tool that enables a user to draw connections between data dimensions and visual variables [5], thus specifying a *visual configuration*. For instance, in the left hand image of Fig. 3, the user connected the longitude with the X axis of the view and the latitude with the Y axis of the view. The user also connected the altitude field of the database to the color of the lines. The resulting connections produce a vertical representation of a one-day traffic record over France (see right Fig. 3). The user can also double-click on axis X or Y of a view to make the field selection menu appear, and change the mapping for that axis.

FromDaDy uses an automatic scaling process to make data visible on the screen. This process is based on scaling with the min/max value of each field of the dataset and the configuration of the view. For instance, the user connected longitude with the X screen and latitude with the Y screen: FromDaDy scales the view so that all latitude and longitude values fit into the view.

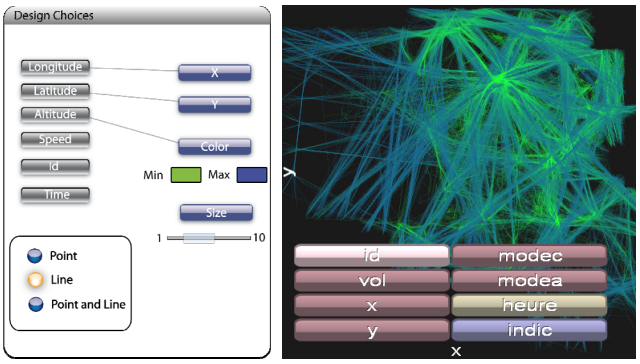



Fig. 3. The connection tool for visual design (left), menu axis (right)

### 5.2 Brushing interaction and incremental selection

The user selects a subset by means of a brushing technique. Brushing is an interaction that allows the user to "brush" graphical entities, using a size-configurable or shape-configurable area controlled by the mouse pointer [4]. Each trajectory touched by the area during the mouse movement is selected, and becomes gray. The selection can be modified by further brush strokes ("Ctrl key" pressed), or by removing parts of it with brush strokes in the "erase" mode ("Shift key" pressed). Our implementation leaves a brush trail, so that the user can see and remember more easily how the selection was made. All trajectories that cross the trail are selected: hence, modifying the selection is like painting or erasing the trail (Fig. 2). While the ctrl and shift key are pressed, the size of the stroke can be adjusted with the mouse wheel. If neither of them is pressed, the mouse wheel

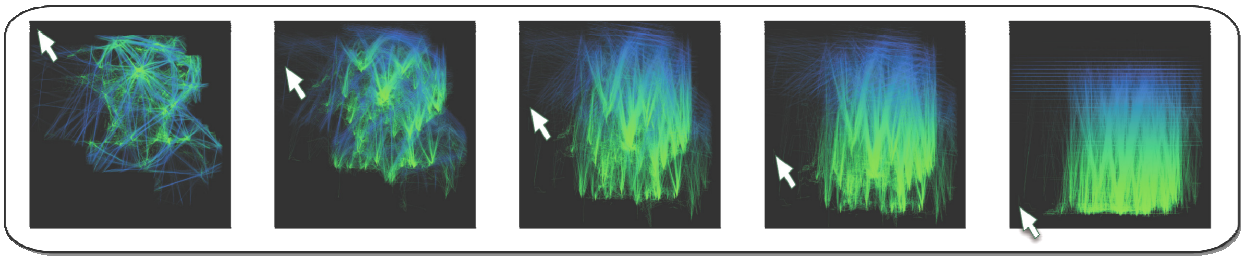


Fig. 4 : Users control the transition between top (Latitude, Longitude) view and vertical (Altitude, Longitude) view by dragging the mouse along the vertical axis.

allows zooming of the view in and out. The combination of fast switching between the *add* or *erase* mode, trail visualization, rapid size-setting, and cursor-centered zooming with the mouse wheel provides for fast, incremental selection.

### 5.3 "Pick and drop" paradigm

Thanks to the brushing technique, the user can select and highlight parts of the displayed data. By hitting the space bar, the user can extract previously selected data and attach them to the mouse cursor (beginning of Fig. 5). By default, the selected data are *picked*: they are removed from the view, and appear in a "fly-over" view (transparent background). When the user hits the space bar for the second time, a *drop* occurs in another view under the cursor. If there is an empty view under the cursor (gray views as shown in Fig. 5), the software creates a new view with the selected data. If the user presses the space bar while moving over an existing view, FromDaDy adds the selected data to this view.

Although it resembles to a regular drag'n'drop operation, we prefer to use the term "pick'n'drop" [14], in the sense that data is removed from the previous view and is attached to the mouse even if the space bar is released.

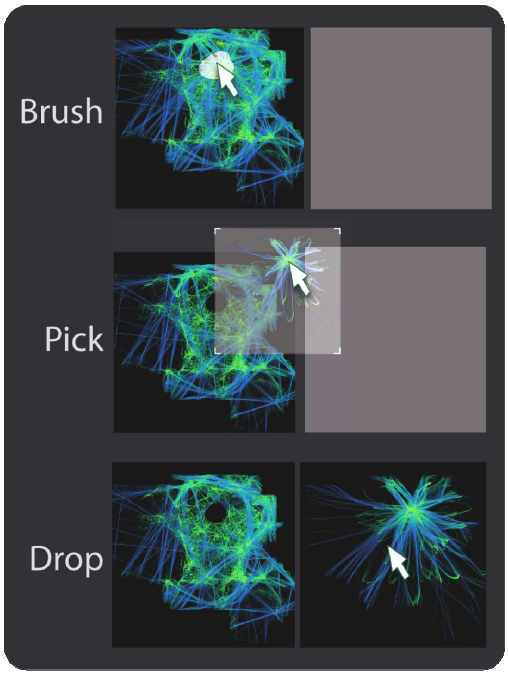



Fig. 5. Pick and Drop interaction

### 5.4 The organization of Views

A session starts with a view displaying all the data. The visualization employs a default visual configuration, i.e. the mapping between data dimensions and visual variables. The view is inside a window, and occupies a cell in a virtual infinite grid that extends from the four sides of the cell. With the brushing and the Pick/Drop paradigm, the user creates new views and changes their visual configurations. The user can select the other cells to display another representation of the data. The user can also destroy a view if the brush selects all the trajectories and if the user picks them.

### 5.5 Rolling dice animation

Sudden changes in the axis of the view are disruptive since they prevent the user from tracking changes over time. Therefore FromDaDy uses an animation similar to "Rolling the Dice" [8]. In other words, one dimension in the focused view is preserved while the other changes. The change is visualized using an animated transition. As in [8], instead of simply interpolating the position of each point for the transition, FromDaDy performs the transition as a 3D rotation. This gives some semantic meaning to the movement of the points, allowing the human mind to interpret the motion as a rotating shape, and to keep the focus on visual entities during the transition. The user can also control the transition with a click and drag along an axis (Fig. 4). Rolling dice animation is also used when dragging the picked data over a view.

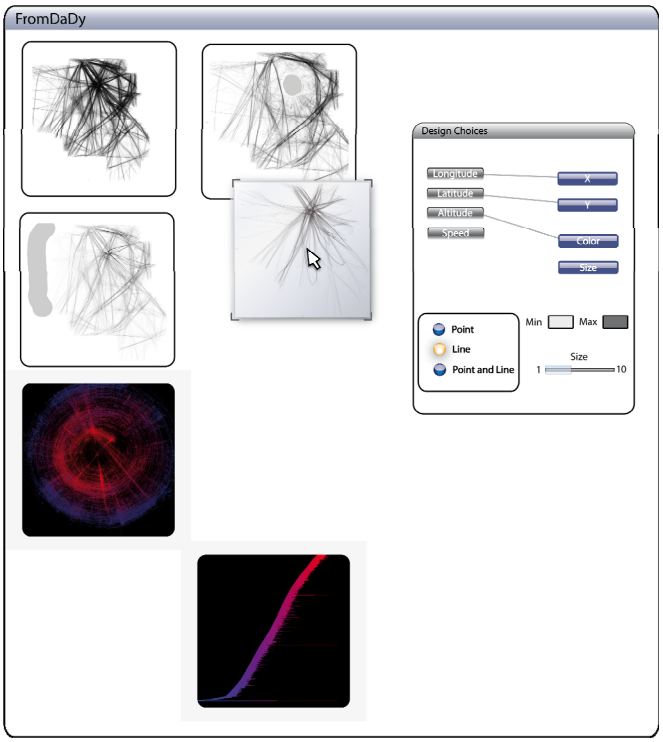



Fig. 6. FromDaDy interface with cells, design tools and one picked selection

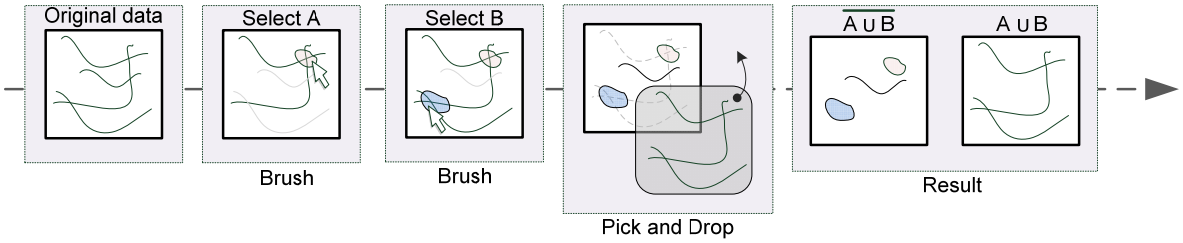



Fig. 7. Union Boolean operation

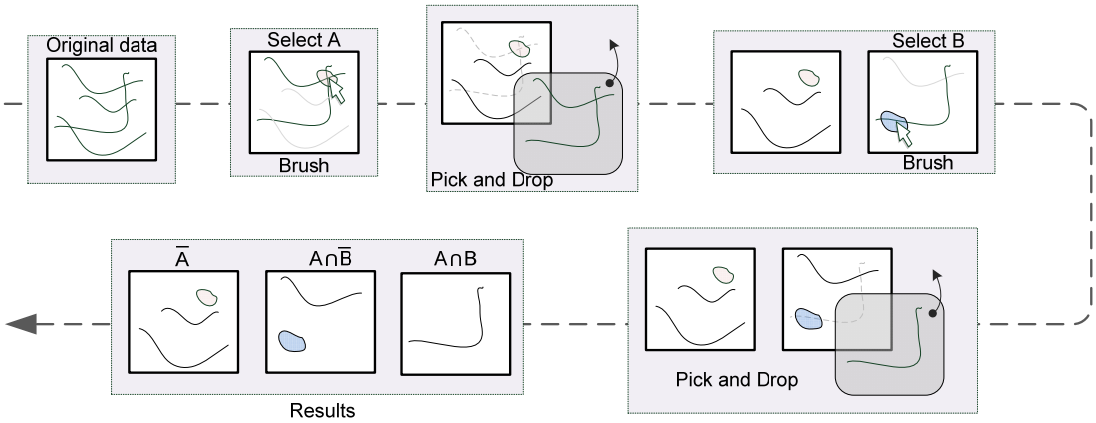



Fig. 8. Intersection Boolean operation

## 6 INITIAL OBSERVED BENEFITS

FromDaDy has been used by engineers and Air Traffic Controllers. During this qualitative evaluation we observe how they took advantage of FromDaDy's assets: the spreading of trajectories across views, the extended features of the pick/drop paradigm, the visual configuration choices, and the implicit Boolean operations.

### 6.1 Spreading data across views

Within FromDaDy, there is a single line per trajectory instance: trajectories are not duplicated, but *spread across* views. The advantage of this technique is twofold. Firstly, it enables the user to *remove* data from a view (and drop them on to the destination view). The fly-over view enables the user to decide rapidly if the revealed data (previously hidden by the picked one) are interesting. Secondly, it makes it possible to build a data subset *incrementally*. In this case, the user can immediately assess the quality of the selection, by seeing it in the "fly-over" view. Furthermore, by removing data from the first view, the user makes it less cluttered, and makes it easier for him to pick data again from the first view and drop them on to the second view.

### 6.2 Picking, transition, and visual configuration picker

The rolling dice animation is also used when the user moves a picked set of trajectories around. When moving over an existing view, the visual configuration of the view may be different from the picked view. In order to prevent sudden changes, FromDaDy animates the transition: the colors, size, pan and zoom change until they reach the parameter of the view under the mouse pointer. This animation is easy to understand and helps the user to figure out the selection layout in the new view *before* dropping. This enables users to re-assess the quality of the selection, as it allows them to forecast the result of the drop. Furthermore, this interaction acts as a visual configuration picker. The user may want to pick trajectories and apply the visual configuration of another view. To do so, the user brushes and picks trajectories, moves the picked trajectories over the view with the desired visual configuration, sees FromDaDy apply the configuration to the picked trajectories, and drops the trajectories into an empty cell.

### 6.3 Line and brush combination for efficient selection

Trajectories are displayed as dots connected by a line. Other design choices may have been envisaged: one color, shape or size per trajectory. Because trajectories are too numerous, lines remain the only suitable design to separate them visually.

As said above, brushing selects entire trajectory instead of single plots. Line brushing has significant advantages: in a very dense area the brushing of a specific trajectory is difficult, whereas the user can select the same trajectory in a less dense area (for example, the surroundings). The zoom is not always a suitable solution to address the problem of selection in a dense area, since the user often needs a complete view on the trajectories. This design choice may lead to false interpretation as the system connects two non-consecutive plots: the line may hide radar detection loss. This kind of data error can be visually detected when trajectories are straight over a long distance.

Trajectory exploration requires more complex selection shapes than a simple rectangle box, and a configurable selection shape, as supported by FromDaDy is more important than, i.e. a movable one. Unlike many visualization systems, FromDaDy employs a simple brushing tool: the user is able to add brush strokes, and remove parts of them. There is no "erase-data" mode, as pick and drop into a trash cell does the same thing. Though simple to master, FromDaDy allows for complex geometrical queries that other visualization software cannot easily perform.

### 6.4 Implicit specification of Boolean operations

Boolean operations are cumbersome to produce, even with an astute interface, as results are difficult to foresee [20]. FromDady reduces this drawback, since any operation of the interaction paradigm (brushing, picking and dropping) implicitly performs Boolean operations. The following two examples illustrate the union,

intersection and negation Boolean operations. With these three basic operations the user can perform any kind of Boolean operation: AND, OR, NOT, XOR…
Fig. 7, the user wants to select trajectories that pass through region A *or* through region B. He or she just has to brush the two desired regions and Pick/Drop the selected tracks into a new view. The resulting view contains his or her query, and the previous one contains the negation of the query. In Fig. 8 the same process is used to find the tracks that pass through A *and* B. By sequencing two "pick and drop" operations, the user formulates his or her request.

### 6.5 Seamless view navigation

FromDaDy gives the user partial control over the organization of the workspace. There are no windows to create and manipulate, and there is only a single layout available (juxtaposed views). This enables quick back and forth pick and drop operations between two views, with rough brushing to unveil hidden trajectories followed by precise brushing to restore some of them. The visual configuration tool is always available and allows for rapid representation change. Hence the user never has to interrupt the exploration process to cope with secondary manipulation.
Furthermore, when exploring a query, the user can arrange the workspace, so as to lay out successive steps. This results in a kind of a storyboard that helps visualize the *procedure* (and not only the data). Thus, in the middle of an unsuccessful exploration, the user can quickly check intermediate views to figure out why the procedure is incorrect.

## 7 SCENARIOS

This section presents two scenarios that underline FromDaDy's assets. This first scenario illustrates how users can explore a dataset and interactively refine their visual queries. The second scenario is a real case, where FromDaDy was used to extract trajectories for a training simulator for Air Traffic Controllers.

### 7.1 Iterative exploration

The visualization shown in Fig. 9 displays air traffic over France during one day. The user wants to display transatlantic aircraft that landed or took off at Roissy airport during one day (Roissy is at the main intersection of the lines). To answer this query, the user first devises a procedure composed of two ordered steps. He or She initially decides to filter aircraft that flew over the Atlantic Ocean. To do so, the user brushes the left hand area of the visualization which selects aircraft that flew over the ocean (Fig. 9, right).

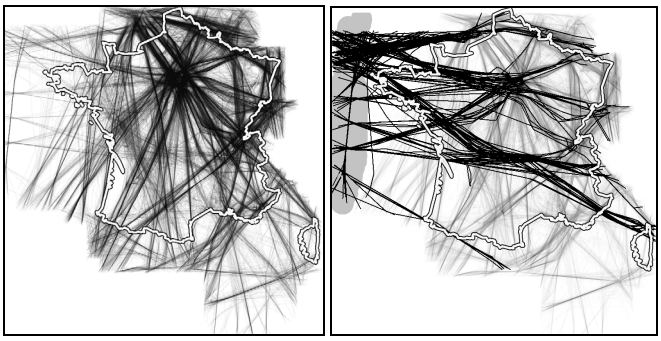


Fig. 9. One day traffic (left), transatlantic selection (right). The thicker polygon is the outline of France.

For the second step, the user changes the view configuration to a vertical view (altitude, latitude) and selects aircraft that have a very low altitude at the latitude of the airport (Fig. 10). The user then changes back the view configuration to a top view (X:latitude, Y:longitude). He or She picks the selected data and starts dragging it. Then the user discovers that trajectories from a second airport, close to Roissy, is part of the selection, and that trajectories landing at Roissy still exist in the view with unpicked data. Furthermore, an intruder aircraft stands out (on the bottom right of Fig. 11). This aircraft performed an unexpected transit flight through Lyon airport, which was not requested.

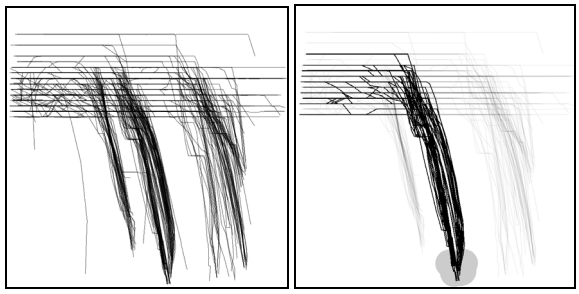


Fig. 10. One day traffic vertical view (left), bottom selection (right).

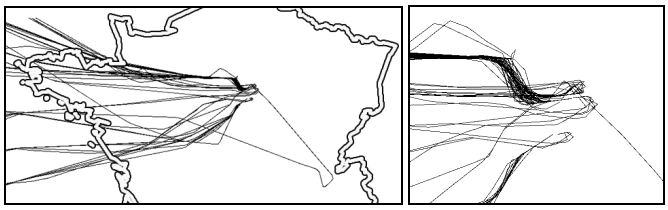


Fig. 11. Resulting selection with one intruder (left), zoomed (right)

The result of the visual selection is effectively inaccurate: the selection misses trajectories that did not end at a low altitude (erroneous data due to radar detection loss or to the 4 minute sampling rate). Furthermore, the vertical view forces the user to select all trajectories with a low altitude at the same longitude of the selected airport (the two main airports in France have the same longitude but not the same latitude).
Hence, the user has to revise the formulation of the query. He or she performs many tentative explorations and finally finds an additional statement: "aircraft that land at this airport do not overshoot it". The user selects aircraft that flew over the ocean, and deletes the ones that overshoot the airport, by using a filter-out brushing operation. He or she thus obtains the required result.

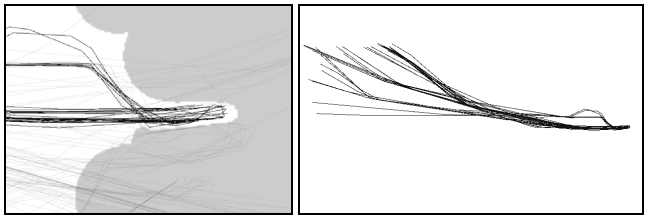


Fig. 12. Selection of non overshooting items (left), Zoom out (right).

This example illustrates how iterative exploration allows the user to find out the correct procedure to use and how the user modified the query to find the correct result. It also illustrates how unexpected data can be easily removed.

### 7.2 Specific trajectory extraction for ATC controller training purposes

In this section, we detail in an actual scenario, in which FromDaDy was used to carry out a data exploration task. The user is a specialist in the Air Traffic Control field. His task was to extract specific aircraft fulfilling the following criteria. Aircraft must pass exactly over specific beacons (corresponding to referenced Flight Routes). Their vertical profile must correspond to a constant climbing

trajectory: there should be no continuous horizontal flight. Finally, aircraft must be sorted by their main departure direction.
Aircraft do not always follow standard Flight Routes. Air Traffic Controllers can shorten a trajectory for optimization reasons. Furthermore, an aircraft can deviate from its trajectory if it overshoots beacons. The user has to filter out this kind of data, even though the criteria that specify them are fuzzy.

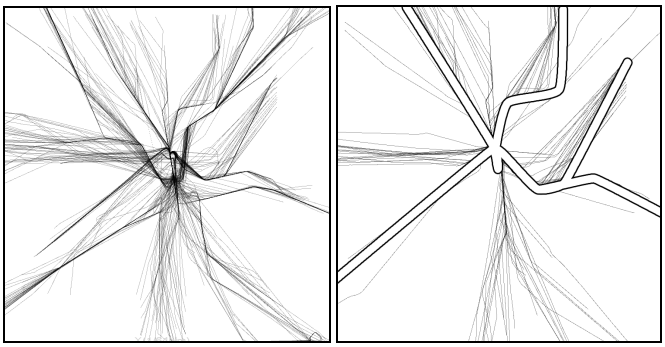


Fig. 13. Original aircraft trajectories (left), landing aircraft trajectories (right) and standard procedures (right hand figure, outlined trajectories are the published Flight Routes by the air transportation authority)

### 7.2.1 Step by step actions

The system first displayed a specific view (longitude->X, latitude->Y) (Fig. 13). As explained above, the data are linked by the "Track ID name", the user can group and join them with a line on the screen, in order to display the different trajectories. Thus, each trajectory concerns a single aircraft.
The user has a rough idea of the position of the standard trajectories and immediately detects them: as they are superposed, they merge into darker lines. The trajectories that surround them are either trajectories shortened by the controller or trajectories that deviated from the initial plan. The user eliminates these trajectories by brushing them and dragging them into a trash cell. FromDaDy also displays two numbers that correspond to the cursor position in the data dimension of the visualization. This enables the user to position the brush precisely at the longitude of the last beacon, and brushes all trajectories that overshoot it, in order to drag them into the trash cell.

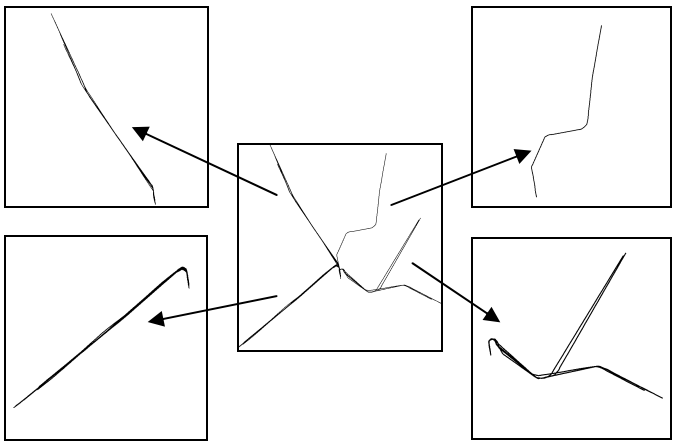


Fig. 14. Trajectories that follows the standard procedures (center), sorted trajectories (corners)

At this stage, the user creates as many views as identified aircraft procedures (two North, one East, and one South departure). To do so, he selects, picks and drops each trajectory into their corresponding view (sorting stage: Fig. 14).
The final step is the selection of the correct vertical profile (Fig. 15). The user changed the visual configuration to a "vertical view" (latitude->X, altitude->Y). The user wanted a constant vertical profile: no aircraft with a continuous flat altitude. Thus the user began to dismiss more aircraft in one view. However, he noticed that he would have been obliged to do so with the three other views. He thus retracted to the previous step by recreating the cell with unsorted trajectories. He applied the vertical profile filtering, and did the sorting step again, thus optimizing his procedure. By organizing the layout of temporary views, the user has been able to target rapidly which steps to retract to.

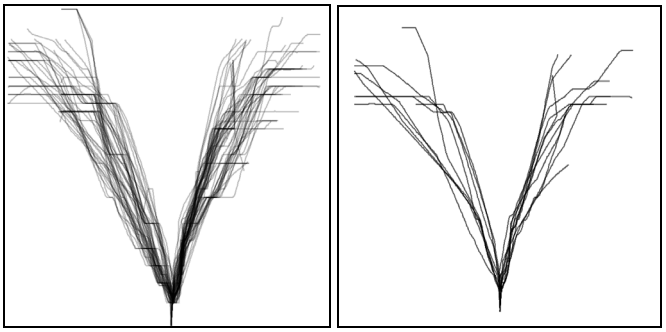


Fig. 15. Trajectories with flat level vertical profile (left), trajectories without flat level (right).

During the vertical profile filtering, the user noticed that the animated transition could have helped him if the views had been correctly arranged. He copied the vertical view under the top view, so that the animation between the top view (longitude, latitude) and the vertical view (longitude, altitude) helped to filter the requested flights: the longitude is common to the two views, therefore the user could focus on the longitude of the last beacon of the Flight Plan and, during the view transition, he could pick out aircraft that had a constant climbing rate up to this longitude (the user can keep the focus on a specific longitude). Again, the ability to organize the workspace rapidly allowed him or her to emphasize the animation feature.

## 8 TECHNOLOGICAL CONSIDERATIONS

FromDaDy is built in C# with the .Net framework 3.0 for interface implementation and DirectX 10 for GPU programming. The brushing technique with 5 millions points is technologically challenging. Therefore we had to take full advantage of modern graphic card features. FromDaDy uses a fragment shader and the render-to-texture technique [10]. Each trajectory has a unique identifier. A texture (stored in the graphic card) contains the Boolean selection value of each trajectory, false by default. When the trajectory is brushed its value is set to true. The graphic card uses parallel rendering which prevents reading and writing in the same texture in a single pass. Therefore we used a two-step rendering process (Fig. 16) : firstly we test the intersection of the brushing shape and the point to be rendered to update the selected identifier texture, and, secondly, we draw all the points with their corresponding selected attribute (gray color if selected, visual configuration color otherwise) (Fig. 16). We also implemented the rendering of points and lines with geometry shaders.
Thanks to these techniques, FromDaDy can display up to 5 million points in real time (frame rates of over 20 FPS) with 2009 computer generation and a 2009 graphic card (8800GTX).

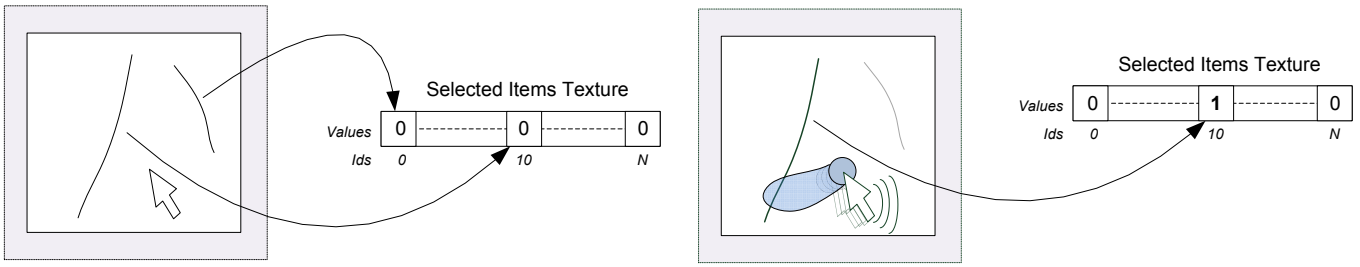



Fig. 16. Brushing technique GPU implementation

## 9 Conclusion

FromDaDy is a multidimensional visualization tool making it possible to explore large sets of aircraft trajectories. FromDaDy uses a minimalist interface: a desktop with a matrix of cells, and a dimension-to-visual variables connection tool. Its interactions are also minimalist: brushing, picking, and dropping. Nevertheless the combination of these interactions permits numerous functions: the creation and destruction of working views, the initiation and refinement of selections, the filtering of sub-datasets, the application of Boolean operations, the creation of relevant steps during the exploration process, and the organization of the desktop layout to create a storyboard and visualize the query building procedure.

Through two scenarios, we showed how FromDaDy supports iterative queries and the extraction of trajectories in a dataset that contains up to 5 million data points with errors and uncertainties. As such, FromDaDy, meets the need for a rapid, flexible and accurate exploration of numerous trajectories in the ATC field.

Our contribution is not a new interaction technique but rather a carefully reasoned justification of how existing techniques can be usefully combined to perform trajectory extraction. The cornerstone of FromDaDy is the trajectory spreading across views with a simple brush/pick/drop paradigm.

We plan to assess FromDaDy with practitioners in traffic analysis. This will enable us to provide longitudinal studies of other tasks. FromDaDy is not limited to displaying aircraft trajectories. It can use different types of data; we plan to perform further experiments with other datasets, such as GPS tracking.

## 10 Acknowledgements

The authors thank Stéphane Chatty, Jean-Luc Vinot, Jean-Daniel Fekete and Pierre Dragicevic for their invaluable discussions and suggestions. The authors add a special thank you to Jean-Paul Imbert, the very first user of FromDaDy.

## 11 References

[1] Ahlberg, C. 1996. Spotfire: an information exploration environment. *SIGMOD Rec.* 25, 4 (Dec. 1996), 25-29.

[2] Ahlberg, C., Williamson, C., and Shneiderman, B. 1992. Dynamic queries for information exploration: an implementation and evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Monterey, California, United States, May 03 - 07, 1992). P. Bauersfeld, J. Bennett, and G. Lynch, Eds. CHI '92. ACM, New York, NY, 619-626.

[3] Baudel, T. 2004. Browsing through an information visualization design space. In *CHI '04 Extended Abstracts on Human Factors in Computing Systems* (Vienna, Austria, April 24 - 29, 2004). CHI '04. ACM, New York, NY, 765-766.

[4] Becker, R. A., Cleveland, W. S., Brushing scatterplots. Technometrics (1987) , Vol. 29, No. 2. (1987), pp. 127-142.

[5] Bertin, J., Graphics and Graphic Information Processing de Gruyter Press, Berlin, (1977).

[6] Card, S.K., Mackinlay, J.D., Shneiderman, B., Readings in Information Visualization: Using Vision to Think. San Francisco, California: Morgan-Kaufmann, (1999).

[7] Derthick, M., Kolojejchick, J., and Roth, S. F. 1997. An interactive visual query environment for exploring data. In *Proceedings of the 10th Annual ACM Symposium on User interface Software and Technology* (Banff, Alberta, Canada, October 14 - 17, 1997). UIST '97. ACM, New York, NY, 189-198.

[8] Elmqvist, N.; Dragicevic, P.; Fekete, J.-D., Rolling the Dice: Multidimensional Visual Exploration using Scatterplot Matrix Navigation Visualization, *IEEE Transactions* on Volume 14, Issue 6, Nov.-Dec. 2008 Page(s):1539 – 1148.

[9] Fekete, J. 2004. The InfoVis Toolkit. In *Proceedings of the IEEE Symposium on information Visualization* (October 10 - 12, 2004). INFOVIS. IEEE Computer Society, Washington, DC, 167-174.

[10] Harris, M. 2005. Mapping computational concepts to GPUs. In ACM SIGGRAPH 2005 Courses (Los Angeles, California, July 31 - August 04, 2005). J. Fujii, Ed. SIGGRAPH '05. ACM, New York, NY, 50.

[11] Hochheiser, H. and Shneiderman, B. 2004. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. Information Visualization 3, 1 (Mar. 2004), 1-18.

[12] Livny, M., Ramakrishnan, R., Beyer, K., Chen, G., Donjerkovic, D., Lawande, S., Myllymaki, J., and Wenger, K. 1997. DEVise: integrated querying and visual exploration of large datasets. *SIGMOD Rec.* 26, 2 (Jun. 1997), 301-312.

[13] Martin, A. R. and Ward, M. O. 1995. High Dimensional Brushing for Interactive Exploration of Multivariate Data. In *Proceedings of the 6th Conference on Visualization '95* (October 29 - November 03, 1995). IEEE Visualization. IEEE Computer Society, Washington, DC, 271.

[14] Rekimoto, J. 1997. Pick-and-drop: a direct manipulation technique for multiple computer environments. In *Proceedings of the 10th Annual ACM Symposium on User interface Software and Technology* (Banff, Alberta, Canada, October 14 - 17, 1997). UIST '97. ACM, New York, NY, 31-39.

[15] Shneiderman, B. 1987. Direct manipulation: A step beyond programming languages. In Human-Computer interaction: A Multidisciplinary Approach, R. M. Baecker, Ed. Morgan Kaufmann Publishers, San Francisco, CA, 461-467.

[16] Stolte, C. and Hanrahan, P. 2000. Polaris: A System for Query, Analysis and Visualization of Multi-Dimensional Relational Databases. In *Proceedings of the IEEE Symposium on information Vizualization 2000* (October 09 - 10, 2000). INFOVIS. IEEE Computer Society, Washington, DC, 5.

[17] Swayne, D. F., Lang, D. T., Buja, A., and Cook, D. 2003. GGobi: evolving from XGobi into an extensible framework for interactive data visualization. *Comput. Stat. Data Anal.* 43, 4 (Aug. 2003), 423-444.

[18] Ward, M. O. 1994. XmdvTool: integrating multiple methods for visualizing multivariate data. In *Proceedings of the Conference on Visualization '94* (Washington, D.C., October 17 - 21, 1994). D. Bergeron and A. Kaufman, Eds. IEEE Visualization. IEEE Computer Society Press, Los Alamitos, CA, 326-333.

[19] Wilkinson, L., The grammar of Graphics. New York: Springer Verlag (1999).

[20] Young, D. and Shneiderman, B. 1993. A graphical filter/flow representation of Boolean queries: a prototype implementation and evaluation. *J. Am. Soc. Inf. Sci.* 44, 6 (Jul. 1993), 327-339.

# Supporting Air Traffic Control Collaboration with a TableTop System

**Stéphane Conversy**[1,2] **Hélène Gaspard-Boulinc**[1] **Stéphane Chatty**[1, 3] **Stéphane Valès**[3] **Carole Dupré**[3] **Claire Ollagnon**[4]

[1]Univ. de Toulouse - ENAC First.Name@enac.fr [2]Univ. de Toulouse - IRIT [3]IntuiLab name@intuilab.com [4]Intactile Design ollagnon.c@intactile.com

**ABSTRACT**

Collaboration is key to safety and efficiency in Air Traffic Control. Legacy paper-based systems enable seamless and non-verbal collaboration, but trends in new software and hardware for ATC tend to separate controllers more and more, which hinders collaboration. This paper presents a new interactive system designed to support collaboration in ATC. We ran a series of interviews and workshops to identify collaborative situations in ATC. From this analysis, we derived a set of requirements to support collaboration: support mutual awareness, communication and coordination, dynamic task allocation and simultaneous use with more than two people. We designed a set of new interactive tools to fulfill the requirements, by using a multi-user tabletop surface, appropriate feedthrough, and reified and partially accomplishable actions. Preliminary evaluation shows that feedthrough is important, users benefit from a number of tools to communicate and coordinate their actions, and the tabletop is actually usable by three people both in tightly coupled tasks and parallel, individual activities. At a higher level, we also found that co-location is not enough to generate mutual awareness if users are not engaged in meaningful collaboration.

**Author Keywords**

CSCW, tabletop, collaboration, air traffic control.

**ACM Classification Keywords**

H.5.3 Information Interfaces and Presentation: Group and Organization Interfaces.

**General Terms**

Design, Human Factors.

## INTRODUCTION

The goal of Air Traffic Control (ATC) is to maximize both safety and capacity, so as to accept all flights without compromising their safety or creating delays. Because air traffic is expected to double by 2030, authorities in Europe and the USA have decided to design new ATC systems. The SESAR [10] and NextGen [6] consortia, both involving billions of euros or dollars, are aimed at changing hardware, software, air space organization and procedures followed by human controllers.

ATC is a highly collaborative activity [1,11]. Collaboration makes controllers more efficient and is essential for safety. The trustworthiness of the global system comes not only from its individual parts (hardware, software or people), but emerges from the process of checking and crosschecking each other's activity. Over the years, various computer systems have been introduced to support ATC activities and controllers were able to use them as a basis for collaboration. However, most recent systems have been largely based on single-person interaction paradigms, and computerization has been obtained at the expense of collaboration. How can designers mitigate this in the systems that are currently being defined?

Recent hardware advances in multi-touch multi-user tabletop systems enable us to imagine potential solutions for collaboration support. Designing such systems requires a deep analysis and understanding of the actual activity to be supported. Even with a sound activity analysis, designers need to find what set of interactions are useful to actually support collaboration: how can tabletop systems improve collaboration compared to other digital systems? How do we maximize users' awareness of what teammates do? How can we enable seamless dynamic task allocation? What set of guidelines should we follow to design effective collaborative tools on tabletop? This paper provides preliminary answers to these questions.

We first summarize the ATC activity and its evolution, so as to clarify the context in which designers work. We then report on an activity analysis focused on collaboration and performed by combining available literature on ATC activities and additional interviews with controllers. From this analysis, we extract a set of requirements and design guidelines. We then present the system and an evaluation.

## AIR TRAFFIC CONTROL AND ITS EVOLUTION

In this section, we briefly outline the tasks of en-route controllers on a typical workstation, and focus on the role of collaboration in traditional ATC workstation designs. We


CSCW 2011, March 19–23, 2011, Hangzhou, China.

then explain why these designs are progressively abandoned and what problems this might pose.

### Traditional en-route ATC tools and procedures

The activity of en-route air traffic controllers consists mainly of maintaining a safe distance between aircraft. To do this the airspace is divided into sectors, each sector being under the responsibility of a team of controllers. When a flight crosses a sector, the controllers guide the pilot by giving clearances (heading, speed, or altitude orders) until the flight reaches an adjacent sector, where other controllers take responsibility for the flight.

In a typical setting, two controllers sit at a Control Position, which is especially designed to support their activities. A traditional Control Position (in France and other countries in Europe) includes a set of vertical screens (the main one being a radar-type visualization), and a horizontal board on which paper flight strips lie [11]. There are two radar screens, one for each controller, often with different configurations (e.g. pan and zoom), and a single horizontal strip board, shared by both controllers. One controller is the *tactical* controller, who gives orders to pilot by radio, and write down his orders on the paper strips. The other controller is the *planning* controller, who is in charge of preparing the newly arriving flights for the tactical controller (possibly by writing notes on the corresponding strips), and of "shooting" exiting flights to other sectors.

### Importance of collaboration in traditional ATC

Past studies have shown that paper flight strips are more than mere information holders and serve as a communication medium [1,5,7,11]: the acts of physically moving, orienting, sticking, holding, and writing on a strip deliver non-verbal messages from one controller to the other. Moreover, as the strips are simply papers on which one can write notes, anyone can interact with them; both controllers can move them and update the information with a regular pen. Other people can also interact with them; for example, in storm situations up to five people might gather at a control position and manipulate the strips. In addition, the flexible co-manipulation of strips enables users to answer very quickly to unexpected events and errors, and enables resilience [14].

Non-verbal communication has been shown to represent up to 50% of all communication acts [2]. Usually, non-verbal communication is done while seeing the teammate and/or the shared environment: physical co-presence enables teammates to use multiple sorts of gestures that improve common understanding of the situation, including deictic gestures, object passing, utterance-like gestures and touching the shoulder to generate awareness [2]. Physical distance between co-workers need not weaken performance in collaborative activities, but it leads them to engage in more demanding communication acts [5,21]. The supplemental work is done at the expense of the main activity, which may be problematic in a situation where work is complex and cognitive load is high. Furthermore, knowledge that one's collaborators know as much as oneself makes the interpretation of collaborators' intentions easier, which in turn makes collaboration better [2,23]. Multimodal communication involving speech and co-located gestures is better at building this mutual knowledge than speech alone [2].

### Automation and its consequences

In order to increase airspace capacity significantly, US and European programs promote *automation* of separation (between aircraft) monitoring and control [6,10]. By delegating the separation assurance function to systems on the ground and in the cockpit, they assume that controllers would shift their attention to such tasks as optimization of traffic flow, or accommodating pilots' requests for route changes. However, the accuracy and efficiency of automated separation depend on the system's up-to-date knowledge of planned and modified trajectories. The current paper and voice-based interaction do not update the system with modifications and orders from controllers, thus preventing the use of automation. This has led to projects that aim at replacing paper and voice with digital tools.

Many software-based systems have been introduced in support of control procedures, including problem management [2], flight lists to partly replace strip boards, etc. However, most introduced systems have used the WIMP paradigm and rely on mouse-based interaction (an exception is [3]), likely because such systems are easy to design and develop. Keyboards and mice are personal devices that are not normally shared: this hinders the ability of a user to interact with her/his teammate's view while the latter is engaged in a conversation on the radio for example. As teamwork is a major asset of previous systems for both safety and efficiency, such individualized tools lower at least efficiency (and some of them have been rejected by users for this reason), and possibly also safety.

## RELATED WORK

A number of research projects have tackled the problem of designing a digital system that can be updated, while preserving collaboration. DigiStrips is a prototype that makes use of two touch screens (one per controller) and finely designed graphics and feedback to support group collaboration [13]. DigiStrips' designers argue that touch screens are appropriate tools to support collaboration:

- They increase mutual awareness. Since touch screens involve gesture, seeing what a colleague is doing with his hand (directly or in peripheral vision) on a touch screen provides many clues on his activity.
- Unlike mice, touch screens are shareable in a fluid manner: a user can interact on his touch screen as well as on his teammate's.

DigiStrips mimics the ability of actual strip boards to lay out the electronic strips so as to convey information. For example, a planning controller may slightly shift or rotate a

strip to the left to make it salient for the tactical controller. Though users could interact with the teammate's screen in DigiStrips, the gap between touch screens prevented fluid passing of objects or the emergence of shared territory [17].

Direct Collaboration interfaces aims at reducing the role of explicit coordination. One strategy of Direct Collaboration is to design interactive objects that serve as a coordination medium [20]. Author proposed three prototypes of interfaces for order preparation and communication. However, they only serve as a demonstration purpose, and were at a too early stage to be tested.

As an alternative to replacing paper flight strips with digital systems, paper strips can be augmented with computing functions. Mackay et al describe how augmented paper strips can provide information to the system, while maintaining paper strips' properties and users' habits [12].

As shown in [8], subtleties in settings can greatly improve collaboration. In an experiment for a new control tool [2], experimenters noticed that a pair of controllers collaborated more when the two radar screens were made closer to one another, and oriented slightly towards the other as opposed to strictly facing the two controllers.

In other domains, numerous systems have been proposed to support close collaboration with tabletops or similar devices (see [18] for a survey on this topic). However, those systems were either a support for a usage or CSCW study, or did not require as precise collaboration as ATC's. Nevertheless, we relied on tabletop design guidelines available in the literature ("System Design" section).

## COLLABORATIVE ACTIVITIES IN ATC

In order to assess the possible benefits of new interaction technologies on ATC collaboration, we need an activity analysis focused on collaboration and how it is supported by traditional tools. A number of studies have been published on the activity of controllers [1,2,5,7,11]. However, practices evolve and subtle differences from past systems may have significant impact on the effectiveness of providing support for an activity. In addition, the available data was not obtained with the exact same purpose of feeding research on interaction design. Therefore, in order to understand current practices we organized four workshops in which we interviewed six different controllers and five ATC experts. During these workshops we aimed at identifying and analyzing situations that involve collaboration in the current French ATC system. We then combined our results with the available analyses to identify the following collaborative activities and situations.

### *Organization and management of flights' lifecycle*

A paper flight strip is the principal embodiment of a flight. Both controllers can manipulate the layout of strips on the board. Layout and orientation changes, hand-written updates to information, and strip disposal are all visible, accountable actions that permit situational awareness to arise non-verbally.

### *Analysis and resolution of problems*

The problem space is always under construction: both controllers are required to perform a "tour of the radar image" or a "tour of the strip board" from time to time in order to discover forgotten actions or unnoticed problems. Working as a pair helps controllers to remember and double-check things to do, and is a cornerstone of safety.

### *Anticipation, preparation, sequencing and sharing of tasks*

When a flight arrives in a sector, the planning controller checks whether it might enter into conflict with another flight in the near future (anticipation). If so, she traces a W on the strip (for "Warning"), and ensures the tactical controller notices the warning when placing the strip on the board (by tapping it with a pen, or by tapping the tactical controller's shoulder). She can also propose changes to flight parameters such as altitude (preparation). Layout on the board can have a variety of significance. For example, flights can be ranked in column by the time of crossing over a beacon: in this case, the planning controller can stack a flight, or insert it in the stack (sequencing). Usually, the tactical controller is in charge of devising a strategy to avoid the potential conflict. However, devising the strategy may be a shared task.

### *Activity allocation*

Activity allocation depends on workload, habits from local culture, and habits arising between the particular pair of controllers. Allocation is always dynamic; no workflow exists that would allow controllers to act in a step-by-step manner, since situations evolve rapidly and allocation requires real-time decision making that is strongly dependant on the current state. Hence, controllers use their tools (radar image, strip boards) more as a whiteboard, on which lie problems to be discovered, problems to be solved, and actions to be done. Actually, part of the activity of a planning controller is to evaluate the status of the other controller in order to devise the best help he can provide. The planning controller constantly adjusts his interpretation of the actions and the state of the other controller. Sometimes, a tactical controller will indicate that the planning controller is wrong in his evaluation, either subtly, or more explicitly (even by shouting at him). The two controllers share responsibilities, but the current paper-based interface does not enforce awareness of responsibility: in fact, responsibility is in users' head and actions, not in the system.

### *Execution and monitoring of actions*

When a flight must turn to follow the planned route, or when the controller has devised an avoidance strategy, the controller needs to give orders to the pilot at the right moment. Hence, part of the activity is devoted to remembering which actions to do at present, or in the near future. Furthermore, resolution of problems depends on the actual execution of orders by the pilots. Hence, controllers must monitor that pilots actually follow orders as given. The planning controller also checks and monitors the

actions of the tactical controller and possibly corrects them in high workload situations.

*Training and high load situations*

Approximately 50% of the time there are more than two controllers on a control position. Often, controllers are apprentices: becoming an expert on a particular sector takes time. During training, the team of controllers includes the apprentice, an expert controller, and a second expert controller to back up the apprentice. In addition, in high workload situation such as storms or emergencies, up to five controllers can gather around the control position to help until the problem is resolved.

## SYSTEM REQUIREMENTS

Based on this analysis of collaborative situations in ATC, we devised a set of requirements for our system. Our primary design goal was to foster seamless collaboration by requiring less explicit communication and fewer coordination acts. Our main assumption is that better collaboration will yield benefits in terms of capacity and safety. More precisely, the system should:

- be updated with controllers' orders. As seen above, this is a prerequisite, and it disqualifies the paper-based system.
- allow more than two users to interact simultaneously with it. This should allow capacity increases since multiple users will be able to handle tasks concurrently. It is also required for *monitoring* and *training & high-load situations*.
- foster mutual awareness. Safety should increase because users will have more means to be aware of teammates' activity and more means to detect problems (*analysis* and *monitoring*).
- foster communication and coordination. This should improve both safety (knowledge of teammate actions) and capacity (less latency). This is required for *organization* and *preparation*.
- foster dynamic task allocation. Capacity should increase because users will be able to pick up new tasks to be done as soon as they have completed existing tasks (*activity preparation* and *allocation*).

## SYSTEM DESIGN

In this section, we describe the various features of our systems. Because of limited space, we focus on features that are explicitly designed to fulfill the requirements, and not the entire system.

In particular, the requirements can be fulfilled if users are aware of tasks to be done, or are able to evaluate workload of their colleague. In addition, it can only be done if any user is allowed to interact with any representation or tools while the other user is engaged in another task. We used a shared, multi-touch, multi-users surface as the basis of our system. Shared surfaces are supposed to exhibit these properties: users are close to each other, and they enable interacting simultaneously if designed appropriately.

**Figure 1: hardware and visualization settings**

### Hardware design

The hardware design is as follows (see Figure 1):

- Two radar displays are presented vertically on the position. These serve as a reference view of the traffic situation and are dedicated to information visualization rather than data input. The radar display is not the focus of the work presented here: it is not touchable and provides for minimal configuration (pan and zoom only).
- A horizontal DiamondTouch (64x48cm) is placed below the radar displays. A projector displays a 1400x1000 image on it. The surface centralizes the input mechanisms, and provides all control tools. More than two controllers can use this shared surface if necessary. We relied on the DiamondTouch ability to identify users, in order to differentiate synchronous interactions [4].

### Representation and Interaction

The horizontal multi-touch screen displays an environment that includes a number of interactive graphical objects. All tools can be seen in Figure 2. We devised the following guidelines to design interactive tools so that they support collaboration:

- Reify actions into objects. Since objects lie on the table, their manipulation may enable accountability [20]; furthermore, they can be passed around and allow for task reallocation.
- Enable partial accomplishment of actions. An action can be separately prepared, checked and accomplished, possibly by different users, thus offering seamless workload allocation.
- Provide as much feedthrough as possible. Since activities must be accountable, it is important that appropriate feedback provide an opportunity for teammates to observe one another's actions.

We also used several guidelines from tabletop and CSCW literature (orientation [9], territoriality [17], tabletop [16], direct collaboration [20] and coupling [22]). In the following, we mention the guidelines that we applied. We

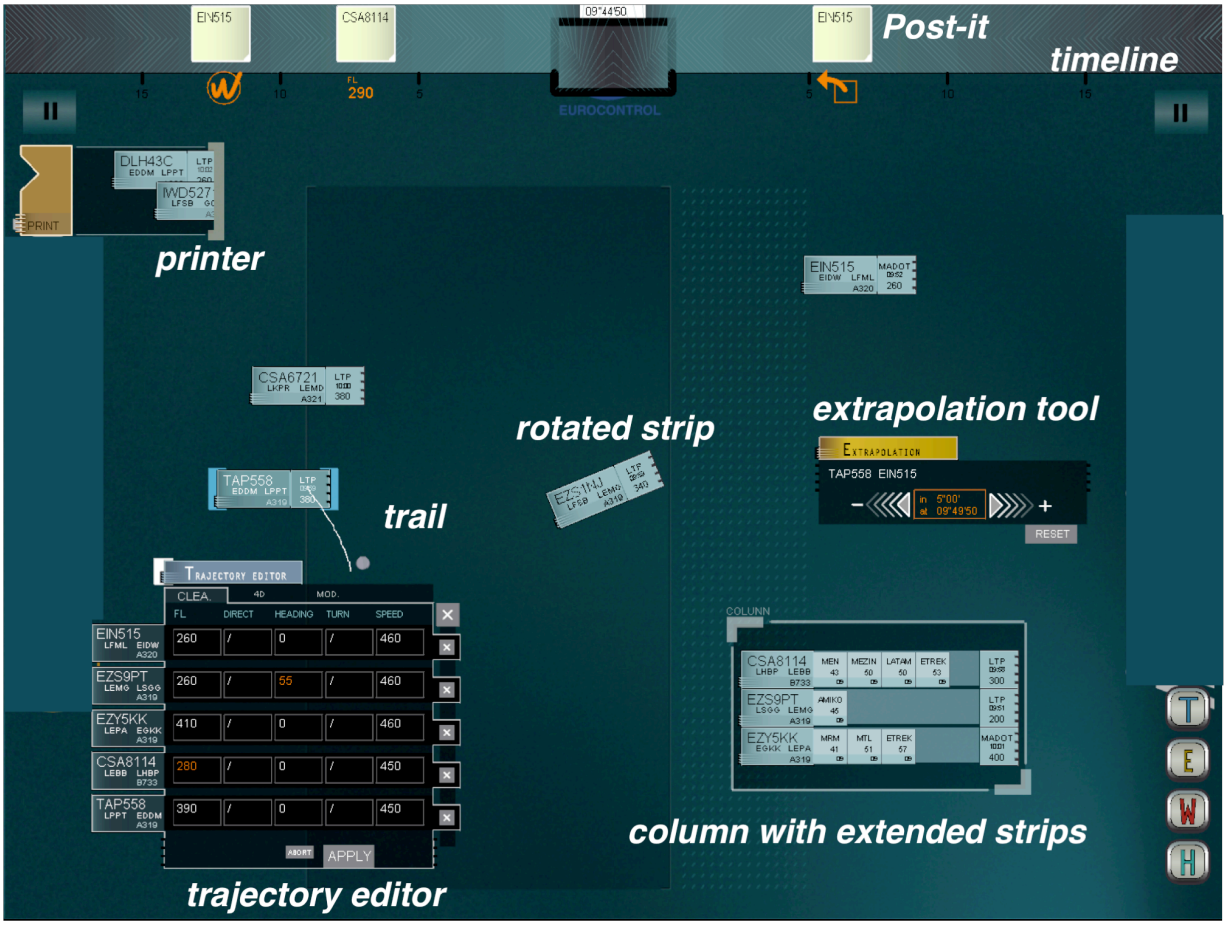



**Figure 2: tabletop view, with interactive objects (named in white/italic)**

chose not to prevent inter-controller conflicts using technical features; instead, we relied on social norms.

*Desktop*

Similarly to the well-known desktop metaphor, the background display is used as a placeholder for other objects. Unlike the radar image, the X and Y dimension of the background has no predetermined semantics; users are free to lay out the objects anywhere on the background. However, users can decide to bring semantics to a specific territory (as discussed in [17], e.g., every entering flight might be placed to the right by controllers) or layout (the top-most flight is the next to enter the sector).

*Strips*

A strip embodies each flight, and displays textual information about it such as call sign, altitude, speed and heading. Strips initially appear in a "printer" box, a metaphor to current hardware. Strips can be dragged and dropped anywhere on the desktop. Users can orient strips non-verbally communicated and provide coordination, as explained in [9]. A column can help organize and manipulate a set of strips as a group; strips inside a column automatically stack onto one another and a strip can be inserted in a column by drag and drop.

*Trajectory editor*

The primary interaction with a strip consists of moving it around. We also chose a spatial model for strip editing rather than a temporal one: in order to edit information on a strip, a controller drops it in a *trajectory editor*. When the drop occurs, a new horizontal tab appears in the editor. Each editable field appears in an edit box: when a field is tapped, a specialized interactor allows for data entry (a radial slider for heading, a vertical slider for altitude, etc.). The trajectory editor fulfills the first requirement (*update the system with orders*).

The trajectory editor lies on the desktop and can be moved around freely for convenience and to allow sharing between multiple users. Edited values are not applied instantaneously: instead, the user must press the "apply" button to confirm changes. Though this seems contradictory to the immediate feedback rule, it allows orders to be prepared and applied later, possibly by another user: this enables users to more finely allocate tasks.

*Extrapolation tool*

The extrapolation tool allows a controller, usually the planning controller, to predict future conflict between two flights (see Figure 3). The tool allows flight paths to be

projected forwards in time, displaying computed future trajectories for selected flights on the two radar images. This provides the tactical controller with an opportunity to be aware of the problems the planning controller is solving.

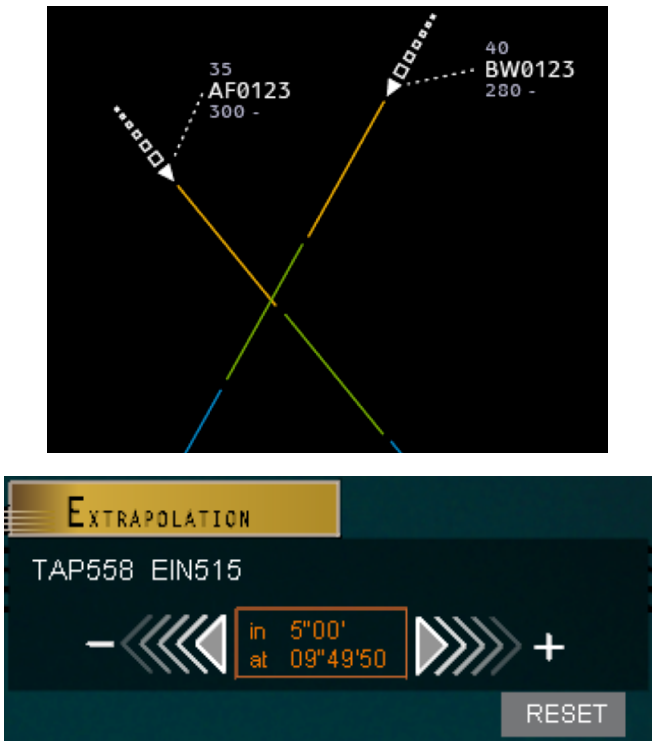



**Figure 3: trajectory extrapolation on radar (top), each colored segment represents the future position in 1 (orange), 2 (green) and 3 (blue) minutes. The extrapolation tool on the table (bottom) controls the amount of future time.**

*Post-it*

Controllers can create Post-its that display a number of call signs and optionally an icon depicting a specific action related to the flights written on the Post-it. To create a special-purpose Post-it, a user triggers the corresponding gesture (S for shoot to next sector, W for warning etc.). Post-its act both as a reminder of actions to do in the future and as a preparation tool similar to the trajectory editor. One controller can prepare an action with a Post-it for another controller to execute later.

*Timeline*

In order to help remember future actions, controllers can place Post-its on a timeline. The timeline is a horizontal strip that lies at the top of the screen. The X dimension of the timeline depicts the time: current time is at the center, and the future extends outwards in both directions, from the center to the edges of the timeline. A ruler that depicts the time according to the X position helps users to position Post-its. Once attached to the timeline, Post-its move automatically towards the center at a pace that follows real time (see Figure 4). As Post-its reach the center, controllers are encouraged to accomplish the associated action, before the Post-its disappear. The double-sided aspect of the timeline enables users to allocate responsibility: each user is responsible for the Post-its that lie on his or her side. Controllers can rearrange the strips, either to specify a different action time or to implicitly redistribute responsibility by moving a Post-it from one side to the other. As Post-its move to the center, they become easier to take from the other controller.

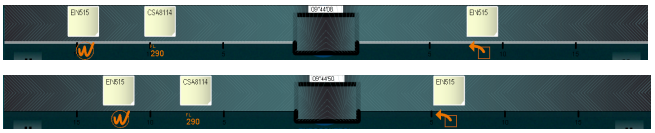


**Figure 4: timeline at two consecutive times: Post-its get closer to the center.**

The timeline and post-it may raise the question of responsibility awareness. We decide not to foster awareness of responsibility, relying instead of the same mechanisms that users employ with the current system. In fact, placing a post-it in the part of the timeline of a controller is of the same nature than placing a paper strip in front of him: nothing will remind a controller to deal with this particular strip, except the other controller.


**Figure 5: Post-its with audio annotation**

*Audio annotation*

As seen above, users can create general purpose Post-its if no specific Post-it applies. In order to remember why they created the Post-it, they can associate an audio message to the Post-it by talking into a microphone while pressing on the "Rec" icon (see Figure 5). Teammates can listen to the message later to be reminded of the action or other contextual information. Controllers can also prepare a vocal order, to be dropped later onto a "radio" object: the audio message is then played on the radio as if the controller were speaking to the pilot. This enables seamless integration of vocal order preparation with current tools and procedures.

*Feedback and feedthrough*

In order to improve situational awareness, the system supports various strategies for making controllers aware of what other controllers do. The system uses direct manipulation, which helps users to understand the actions of others since each action requires gestures and time to accomplish [19]. Controllers' attention is divided between

the vertical radar screen and the horizontal screen. Feedback displayed onto each screen is translated appropriately onto the other: for example, touching a strip highlights the corresponding representation on the radar image. This allows controllers to be aware of each others' actions even while looking at the radar. Additionally, any touch interaction on the surface leaves a trail on the surface that gradually disappears (see Figure 6). This allows a controller who looks elsewhere to get an idea of what has been done when his attention returns to the table. Finally, all actions use smooth animation to depict state transition, which helps users notice changes made by their colleagues [15]. For example, inserting a strip in a column makes the other strips separate smoothly to make room.

**Figure 6: touch gestures leave transient trails**

## PRELIMINARY EVALUATION

We have conducted four pilot studies to evaluate our design choices. The studies were qualitative and involved a limited number of subjects and trials. As such, they yielded preliminary results only; however, we did make several useful observations. The studies were not meant to test whether our system is better than current systems in terms of capacity or safety. Rather, they test to what extent the requirements we listed above (orders notification, more than two users, mutual awareness, communication and coordination, dynamic task allocation) are fulfilled.

### Participants, settings and procedure

The study participants were three air traffic controllers and five ATC experts. Study 1 involved four groups of two, Studies 2 and 3 two groups of two, while Study 4 involved the three current ATC controllers only. We used specialized software that replays recorded air traffic in real-time. The radar display and the tabletop view display the traffic by "listening" to the replay software. The replay software is able to modify the simulated traffic according to orders given by users. The setting is shown in Figure 7.

In addition to direct observation, we videotaped the sessions with two cameras: one with a large field of view, to film the whole setting (people, horizontal surface, and vertical radar screen) and to catch any interaction between people, and one close to the multi-touch surface to catch gestures and the interactive environment.

After we provided a general introduction to the system, the participants were allowed to interact with it. They performed the main possible interactions in order to discover and learn how to interact with the system. Once they appeared familiar with the system (~10min) they ran through each study, which consisted of reading instructions and fulfilling a set of tasks. We also ran a discussion with subjects after each study.

### Study 1: mutual awareness

The main objective of the first study was to evaluate how the interactive surface affects the awareness of each other's action. The two controllers each had a list of six actions to perform. After completing the scenario, each controller was asked to describe the actions performed by the other. Four groups of two controllers performed this test.

The results were identical for all four groups: no controller was able to describe any action performed by the other. This can be explained by two observations. First, as we noticed in the video, subjects were still performing as beginners and spent a lot of time and cognitive resources discovering how to interact with the table, at the expense of mutual awareness. Later studies benefited from this learning process; however, this study was negatively impacted. Second, the actions required were not embedded in a real activity and were not strongly related to one other, making them less "guessable" by a colleague.

Hence, study 1 did not show that our system supports mutual awareness. However, it does illustrate that proximity is not necessarily sufficient for awareness of other participants' actions: context and engagement in a meaningful collaboration is also important.

### Study 2: communication

The main objective of our second study was to evaluate how the interactive surface might facilitate collaboration between the two controllers, through the different artifacts provided. Three scenarios were exercised during the test.

#### *Scenario 1: non-verbal communication*

In the first scenario, the tactical controller was asked to give clearances to aircraft via orders over the radio, and to update the system using the trajectory editor. In parallel, the planning controller was asked to integrate new flights by dragging the flight strips from the printer box to the appropriate column. Then, the planning controller was asked to focus the attention of the tactical controller on a conflict. We instructed the subjects that they could use any features afforded by the system (Post-it, orientation, timeline) to accomplish their tasks but that they were not to speak to one another. In practice, the ability for the planning controller to communicate with the tactical controller silently is important since the tactical controller may be speaking to pilots by radio.

**Figure 7: Multiple users engaged in the task and interacting simultaneously**

Planning controllers from both groups used Post-its associated with the two flights in conflict as communication artifacts. They dragged the Post-it into the timeline on the tactical controllers' side. The planning controller of the second group additionally placed the associated flight strips beside the tactical controller's column, in order to make later processing easier. While the first group succeeded in communicating, the second group failed, despite the additional step. Two observations may explain this result. The first group was composed of two current controllers. Before alerting the tactical controller, the planning controller actually checked whether there was a potential conflict. To do so, he used the extrapolation tool, and verified on the radar screen where and when trajectories cross. Feedthrough on radar allowed the tactical controller to notice planning controller's actions. This helped the tactical controller build mutual awareness, and made her more eager to pay attention to potential problems. Both controllers confirmed this during debriefing interviews.

The planning controller of the second group, who was not a current controller, selected the two flights randomly. This provided insufficient feedthrough to support the tactical controller (a current controller), who in turn was not engaged enough to really pay attention to a fake problem. It is probable that two current controllers are more used to paying attention to each other's actions. However, the efficiency of tools in supporting collaboration is highly dependent on whether or not the controllers share the same skills, knowledge and training [2].

*Scenario 2: verbal communication*

Scenario 2 was similar to scenario 1, except that verbal communication was allowed. The planning controller was asked to integrate new flights and to realize an extrapolation on two flights. In parallel, the tactical controller was asked to call the planner's attention on a flight in order to initiate a route negotiation with an adjacent sector.

The two groups chose the same strategy to achieve this goal successfully: the tactical controller took the flight strip, placed it under the planning controller's eyes, and talked to him while pointing at the label. In both groups, the planning controller understood immediately what to do.

It is interesting to underline that this is the current means of collaboration between French air traffic controllers: they use the paper flight strips to enhance the efficiency of verbal communication and to eliminate ambiguity about involved flights. This property of a single physical flight representation has disappeared in some new systems where each controller has his own screen to display flight plan information. The shared surface restored the flight representation as a coordination object.

**Study 3: coordination**

The aim of Study 3 was to evaluate the efficiency of the Post-it as a mean for coordination. The tactical controller was asked to give clearances and to update the system, using the trajectory editor. In parallel, the planning controller was asked to edit a Post-it on a flight, in order to notify the tactical controller of a "frequency change."

Despite the Post-it motion executed by the planner on the tactical controller's side of the timeline, neither of the two tactical controllers noticed the Post-it. It appears that the topological configuration makes it difficult to share information between seated users: as the timeline lies at the top of the interactive surface, it is out of the visual field and difficult to reach when one is seated.

**Study 4: more than two users and dynamic task allocation**

The aim of our fourth study was to assess the effectiveness of the system in supporting collaboration in situations involving more than two users, such as training or storms. The study involved real controllers only: two of them were asked to do a regular air traffic control using the tabletop system. After five minutes, a third controller (the "supporting" controller) was asked to help the others to control traffic (see Figure 7).

*First period: two controllers*

We made the following observations of the first period:

- Controllers moved the printer box, initially placed on the left top of the interactive surface, to a location between them. In this position, both controllers can access the printer box and integrate new flights. This illustrates the ability to configure the environment so as to foster collaboration.
- Flight strips on the surface were actively used to highlight and locate flights on the radar image. Hence flight information served as an individual aid (which may improve mutual awareness), as well as communication support (highlighting a flight for each other).

*Second period: Three controllers*

We asked the third controller to come and help due to a peak of traffic. She stood behind the two principal controllers and began to manage new flights that had piled up in the printer box. The three controllers then succeeded in working together on the surface (Figure 7). Several situations were observed:

- A situation consisting of close collaboration between the tactical and supporting controller, using the flight strip combined with the radar image;
- A situation consisting of parallel activities, where the tactical and supporting controller worked on one problem while the planning controller worked on another.

Controllers succeeded in allocating tasks to the newly arrived controller. They did it in a fluid manner thanks to the capability of organizing the flight strips on the horizontal surface. It enabled a natural division of the surface into private spaces for each controller and common spaces used for exchanges (findings similar to [17]). However, during the debriefing session, the main negative feedback from the controllers was related to the surface size: it appeared too small when used by three people.

## DISCUSSION

Our evaluation was partial: we did not test every feature of our system. For example, in our trials no subject used the audio annotation. We do not know whether the fading visual trails helped controllers with mutual awareness. Testing this feature would be hard in practice since it would require a scenario in which a peripheral system would distract one controller such that she has to understand what she missed during the disturbance. Even if she did not appear to benefit, this would not allow us to infer that such a feature is useless. Longitudinal studies would be better to really assess it. However, we learned a number of lessons through these studies.

Subjects did not make a heavy use of orientation. This may be due to the complexity of our rotation interaction: one has to press, wait half a second to enter the rotate mode, then rotate the strip. This finding is consistent with guidelines for rotation [9]: we failed to provide a fluid interaction for rotation, which prevents actual use (the device we used makes it difficult to implement a multi-touch multi-user system without a temporal mode). Similarly, the position of the timeline was not adequate, which is consistent with previous work on territoriality [17].

Another finding of this work is that activity knowledge and engagement are very important for the assessment of the effectiveness of the tools. In fact, we "forced" collaboration for the sake of study 1. This led the two subjects to execute unneeded collaborative tasks and prevents the formation and assessment of mutual awareness. In study 2, actual controllers succeeded in building mutual awareness because they were engaged in a meaningful collaboration and they shared a common understanding of the situation.

We found that tabletop effectively supports communication and coordination: users were able to communicate verbally and non-verbally by using gestures and territories. Feedthrough plays a role in building mutual awareness. For example, the extrapolation tool helped the tactical controller understand what the planning control was doing. Highlights on radar screen also helped users gaining an idea of their teammate's actions. The fact that mutual awareness is key to safety in critical systems highlights the importance of good feedthrough.

Study 4 (with 3 controllers) gave very interesting results. The controllers had manipulated the system during the previous studies, and they were at ease in using it in real-time conditions. Moreover, we observed that users were really engaged in their task: the flow of action was very smooth, since the interface allowed multiple controllers to manipulate it at the same time. Users were able to dynamically allocate tasks, and engage in tightly coupled or parallel tasks. This truly illustrates what we expect from such a system: that appropriate, seamless technologies and tools make collaborative activities such as air traffic control smooth and efficient.

During the debriefing and discussions we had with the participants, they made remarks about possible benefits of the system. For example, in storm situation the combination of multiple people and multiple concerns often leads to contradictory actions. Even if the reified actions are not a complete description of the strategies involved, a trained controller can infer the appropriate action from the available information. The training process might also benefit from our system. At the beginning of ATC training, apprentices practice in simulation under observation by instructors. As in the storm situation, the reified actions and the timeline might help the instructor better understand the apprentice's strategy. The instructor can then revise apprentices' priorities (the major difficulty faced by apprentices) by rescheduling planned actions on the timeline while explaining the corrective actions to the apprentices by showing or tapping on other reified actions (i.e. speechless explanation). After the simulation ends, another tool could replay the actions performed by the apprentice. According to the instructors, "such a tool would be invaluable". During actual traffic control, an apprentice could prepare and apply actions that an instructor would in turn validate in order to execute them effectively.

## CONCLUSION

We have described a complete example of a digital tabletop system designed for ATC, a real-world, complex task environment. We designed the system to circumvent flaws associated with traditional technology in the context of a highly cooperative activity. We based our design on an analysis of ATC controllers' activity, with a focus on collaboration, and provided a set of requirements (support more than 2 users, mutual awareness, communication and coordination, task allocation). We devised a set of

guidelines to design our system (reification, partial accomplishment of actions, feedback), and presented a set of new interfaces and interactions. Finally, the paper provides initial data of an exploratory evaluation with ATC experts on the effectiveness of the specific interface design features included into the system. Researchers and practitioners can use the design guidelines as is, and get inspiration from the artifacts. They can also gain some insights into the utility of the specific interface design concepts in this, and potentially other complex, collaborative task domains.

We obtained mixed results with the evaluation, a typical outcome of non-tightly controlled experiments: fluidity and dynamic repartition are largely unpredictable (it depends heavily on the particular pair of users for example), and thus difficult to control. Nevertheless, pilot studies show that our tools partially fulfill our expectations, and give insight on future evaluation or ideas. Assessing the effectiveness of our tools requires more than a few studies (new digital systems for ATC have been in the design phase for 20+ years because assessing them is so difficult). As tabletop technology matures, more accurate and reliable systems can benefit from the work presented in this paper. Together with longitudinal studies with reliable systems, it can provide convincing arguments to the introduction of tabletop based systems in real-world, critical activities.


## ACKNOWLEDGMENTS

Funded by Eurocontrol under grant EEC A06/12013BE. We thank V. Peyruqueou, J. Viala, H. Hering, M. Brochard and ATC experts who participated in the design, G. Philips, N. Roussel and the reviewers for their helpful comments.


## REFERENCES


1. Bentley, R., Hughes, J. A., Randall, D., Rodden, T., Sawyer, P., Shapiro, D., and Sommerville, I. 1992. Ethnographically-informed systems design for air traffic control. In *Proc. of CSCW '92*, ACM, 123-129.
2. Bressolle, M.-C., Pavard, B., Leroux, M. The Role of Multimodal Communication in Cooperation: The Cases of ATC. *LNCS* (1374), 326-343, Springer, 1998.
3. Chatty, S. and Lecoanet, P. 1996. Pen computing for air traffic control. In Proc of CHI96, ACM, 87-94.
4. Dietz, P. and Leigh, D. 2001. DiamondTouch: a multi-user touch technology. In *Proc of UIST '01*, 219-226.
5. Dumazeau, C., Karsenty, L. Communications distantes en situation de travail : favoriser l'établissement d'un contexte mutuellement partagé. In *Le Travail Humain*, (PUF), Vol. 71 N. 3, p. 225-252, 2008.
6. Erzberger, H. Transforming the NAS: The Next Generation Air Traffic Control System. NASA/TP-2004-212828, Oct. 2004.
7. Hughes, J. A., Randall, D., and Shapiro, D. 1992. Faltering from ethnography to design. In *Proc. of CSCW '92*. ACM, New York, NY, 115-122.
8. Ishii, H. and Kobayashi, M. 1992. ClearBoard: a seamless medium for shared drawing and conversation with eye contact. In *Proc. of CHI'92*, ACM, 525-532.
9. Kruger, R., Carpendale, S., Scott, S. D., and Tang, A. 2005. Fluid integration of rotation and translation. In *Proc. of CHI '05*. ACM Press, 601-610.
10. Ky, P. and Miaillier, B. 2006. SESAR: towards the new generation of air traffic management systems in Europe. In *ATCA Journal of ATC Quaterly*, 14(1).
11. MacKay, W. E. 1999. Is paper safer? The role of paper flight strips in air traffic control. *ACM Trans. Comput.-Hum. Interact*. 6, 4 (Dec. 1999), 311-340.
12. Mackay, W. E., Fayard, A., Frobert, L., and Médini, L. 1998. Reinventing the familiar: exploring an augmented reality design space for air traffic control. In *Proc. of CHI 1998*, ACM, 558-565.
13. Mertz, C., Chatty, S., Vinot, J.-L., Pushing the limits of ATC user interface design beyond S&M interaction: the DigiStrips Experience, *Proc. ATM'2000 R&D seminar*.
14. Rognin, L., Salembier, P., Zouinar, M. Cooperation, reliability of socio-technical systems and allocation of function. 2000.*Int. J. Hum.-Comp. Studies*, 52, 357-379.
15. Schlienger, C., Conversy, S., Chatty, S., Anquetil, M., Mertz, C. Improving Users Comprehension of Changes with Animation and Sound: an Empirical Assessment. In *Proc. of Interact 2007, LNCS*, 207-220. Springer.
16. Scott, S.D., Grant, K.D., and Mandryk, R.L. System guidelines for co-located, collaborative work on a tabletop display. *Proc. ECSCW'03*, Springer, 159-178.
17. Scott, S. D., Carpendale, S. and Inkpen, K. M. 2004. Territoriality in collaborative tabletop workspaces. In *Proc. of CSCW'04*. ACM, 294-303.
18. Scott, S. D., Carpendale, S. Eds. Interacting with Digital Tabletops. In *Special Issue of IEEE Computer Graphics and Applications*, IEEE 26(5), 2006.
19. Shneiderman, B., Direct Manipulation: A Step Beyond Programming Languages, *IEEE Computer,* 16(8), 1983, 57-69.
20. Sire, S., Chatty, S., Gaspard-Boulinc, H., Colin, F.-R. 1999. How can groupware preserve our coordination skills? Designing for direct collaboration. In *Proc. of Interact 99*, IFIP, pp. 304-312.
21. Sperber, D., & Wilson, D. La pertinence. *Communication et cognition*, Les éd. de minuit, 1989.
22. Tang, A., Tory, M., Po, B., Neumann, P., and Carpendale, S. 2006. Collaborative coupling over tabletop displays. In *Proc of CHI'06*. ACM, 1181-1190.
23. Wilkes-Gibbs, R., and Clark, H. Coordinating beliefs in Conversation. *Journal of Memory and Language*, 31, 1992,183-19

# Augmenting the Scope of Interactions with Implicit and Explicit Graphical Structures

**Raphaël Hoarau**
Université de Toulouse, ENAC, IRIT
7 av. Edouard Belin, Toulouse, France
raphael.hoarau@enac.fr

**Stéphane Conversy**
Université de Toulouse, ENAC, IRIT
7 av. Edouard Belin, Toulouse, France
stephane.conversy@enac.fr

**ABSTRACT**
When using interactive graphical tools, users often have to manage a structure, i.e. the arrangement of and relations between the parts or elements of the content. However, interaction with structures may be complex and not well integrated with interaction with the content. Based on contextual inquiries and past work, we have identified a number of requirements for the interaction with graphical structures. We have designed and explored two interactive tools that rely on implicit and explicit structures: ManySpector, an inspector for multiple objects that help visualize and interact with used values; and links that users can draw between object properties to provide a dependency. The interactions with the tools augment the scope of interactions to multiple objects. A study showed that users understood the interactions and could use them to perform complex graphical tasks.

**Author Keywords**
Graphical Interaction Design, Instrumental interaction, Exploratory Design.

**ACM Classification Keywords**
H5.2 [Information interfaces and presentation]: User Interfaces: Graphical user interfaces - Interaction Styles.

**General Terms**
Design, Human Factors.

## INTRODUCTION

When using computerized tools such as real-time editors, presentation software, GUI builders, etc. users create and manipulate graphical objects on the screen. They can edit them individually, e.g. change their color or their stroke width. Users can also consider and interact with sets of objects as opposed to individual objects. To do so, they may be required to structure the scene, by relying on concepts such as groups, styles, or masters. According to the Oxford dictionary, a *structure* is "the arrangement of and relations between the parts or elements of something complex". Using a structure may have multiple assets, such as helping users conceptualize the scene they are creating ("the background of the slide includes this drawing and this text", "this set of slides is a subpart of the presentation" etc.), and think better about the problem at hand. Here, we are interested in structures as means to interact with the content: since structuring involves sets of objects, the actions done on an element of the structure may have an effect on several objects at once.

In current interactive systems, the use and the management of structures may be complex. Users have to create and maintain them. Depending on the kind of structure, some operations may be cumbersome or impossible to do, which prevents users to explore the design space of their particular problem. Furthermore, systems that provide structuring do not leverage off the structures fully to provide users with new ways of interacting with the content.

Interactions with structure and with multiple objects through a structure have not been studied extensively in the past. Of course, a number of past works have identified the problem [6], but few concepts or properties targeted it explicitly [2,12]. For example, what are the interactions that enable users to define sets of objects? What are the available means to augment the scope of interaction i.e. apply an interaction to several targets? What are the concepts that may guide the design of such interactions?

The work presented in this paper aims at improving the management of structures as means to augment the scope of interactions. Based on contextual inquires and related work, we present a number of requirements pertaining to the interactions with structures. We then present two interactive tools that aim at fulfilling those requirements. The first one is ManySpector, an inspector for multiple objects. ManySpector displays all used values for a property given a set of differing objects, whereas a traditional inspector displays no value. This reveals an *implicit* structure of graphics (the sets of objects that share a graphical property) and offers new interaction means. The second one is based on links that users can draw between object properties to provide a dependency. The resulting property delegation graph is a means for users to provide an *explicit* structure. We then report on a user study involving those tools.


*CHI 2012,* May 5-10, 2012, Austin, TX, USA.

## CONTEXTUAL INQUIRIES AND SCENARIO

We have based our work on concrete and realistic case studies. We have conducted five contextual inquiries with "designers", the design activity being taken in its broadest sense: edition of graphics (Illustrator and OmniGraffle), courses schedule (iCal), architecture (Auto-CAD), or lecture presentation (PowerPoint). We have written a dozen scenarios that describe accurately the activities.

In order to introduce the problem, we present one of the scenarios. This scenario illustrates a number of requirements pertaining to interactions on several objects, with or without a structure. The scenario is real but adapted slightly for illustration purpose: some interactions that are deemed as impossible (e.g. with Inkscape) might be possible with other tools (e.g. with Illustrator and vice-versa). The steps are annotated in italic to characterize them. We detail the annotations later in this section.

Elodie is a designer tasked with creating the graphics of a custom software keyboard for a tablet computer. Using a graphical editor, she creates a first key. She draws a rounded rectangle with a solid white fill and a surrounding stroke. She adds a rectangle inside the previous one, with a blue gradient fill (no stroke). She selects both rectangles with a selection lasso (*designation*) and groups them with a command in a menu (*structuring*). She then adds a soft shadow effect on the group. She overlays a label with a text 'A' on the group of rectangles and centers the label and the group by invoking a 'center' command on a toolbox. She then forms another group with the label and the groups of rectangles, and names it "key" in the tree view of the graphical scene provided by the application (*structuring*). This first key serves as a model to create other keys: she duplicates the key, and applies a horizontal translation to the copy. She proceeds with this action several times in order to get a row of keys (Figure 1). She then modifies the text of each key one by one (Figure 2).


**Figure 1. The user creates a key, and duplicates it.**


**Figure 2. The text of the 'I' key is not centered.**

When she changes the letter 'A' for 'I', she realizes that the 'I' text is not centered with regards to the rectangles (Figure 2). The first object was specified incorrectly: if the three objects (label, gradient rectangle, rectangle) are correctly aligned, the text of the label is not centered. The problem was not noticeable with the first letters (AZERTYU) since their widths are similar. Each label being in a heterogeneous group (containing object types other than label), the system does not provide a text center command that can be applied to a selection of objects. She has to click multiple times on an object to reach the label and apply the 'text centered' command. Therefore, she estimates that it is more efficient to start over: she deletes all copies, ungroups the first key, centers the text, groups the objects again, copies and moves the copies, and modifies each letter one by one.

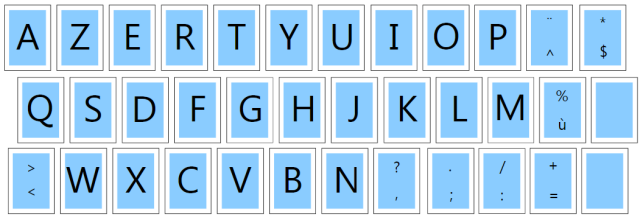



**Figure 3. The entire keyboard with the double keys.**

Elodie has finished the entire keyboard. Some of the keys are double keys that contain two smaller labels at the top and the bottom of the key (Figure 3). She wonders whether the double key labels are too small and she wants to explore new sizes (*exploratory design*). First she has to find each double key in her design (*searching*). To do so, she zooms out to make the keyboard entirely visible. This allows her to identify each double key. Again, she has to change the size of the labels one by one.

The scenario illustrates several requirements.

*Structuring* Elodie relied on the ability of the system to allow creation, modification, and management of sets. For example, she created a single group with two rectangles, then another group with the previous one and the label.

*Designation* Elodie designated objects, properties and actions. For example, she changed the "alignment" property of the label to "centered".

*Scope of actions* Elodie acted on multiple objects at once. For example, she grouped objects because she wanted to consider them as a single entity that keeps the relative positions between subparts, but also because she wanted to apply a single translation on three objects at once. Conversely, she was not able to apply the command 'set alignment' to several objects at once.

*Seeking* Elodie needed to retrieve objects: she had to search objects whose content is similar to other ones. The search action requires visually scanning the graphical objects and seeking candidate objects, at the risk of forgetting some of them. The more the objects, the more difficult it is to find out particular ones, especially if the features to search for are not pre-attentive [4]. As the number of keys increases, each modification gets more costly, not only because of the number of actions to repeat, but also because of the required visual search effort.

*Exploratory Design* Elodie explored parts of possible solutions, and modified existing parts of solutions. By combining action, visualization of intermediate results and thinking, she co-discovered the problem and the solution. In

doing so, she was pursuing an exploratory design activity. This phenomenon is important for activities in which the expected result is not known in advance: graphics edition activities, slides design, or class hierarchy design [8][24].

## RELATED WORK

Past works have tackled the problems of managing structures, and interacting with multiple objects, either explicitly or implicitly. We present them along three axes: interactions for structuring the content provided by interactive systems, design and evaluation of interactions for structuring, and structuring in programming.

### Structuring for users

*Groups* Traditional graphical editors allow users to create groups from a set of objects previously selected by the user, and to act on those groups. The only operation available for a group is 'ungroup', which removes the group entity and selects all objects that were part of the groups (no modification, addition, or subtraction). Selection can be seen as a transient group, with 'add' and 'remove' operations by holding the shift key and selecting several elements, or holding the ctrl key and clicking on individual elements. Some tools support heterogeneous settings, but with specific properties only *e.g.* translation, scale and rotation: all elements in the group are transformed accordingly. Conversely, some operations (e.g. 'set color') cannot be applied to groups, supposedly because some elements inside the group do not "understand" them. This forces the user to ungroup and apply the command on each object. In this case, interaction with the structure is not well integrated with interaction with the content.

*Trees* Groups can be part of a surrounding group, turning them into trees or hierarchies. Support for management of such hierarchy ranges from no support at all, to navigation in the hierarchy of parents [18], and tree views in structured graphics editors (e.g. Inkscape or Illustrator). A tree view enables users to reparent elements with a drag and drop. However, there is no support for other operations, such as applying a color to a node in order to change all children.

*Masters* A Master is an element used as a "model" for other elements. For example, PowerPoint enables users to define in a master slide the appearance that other slides would inherit. Sketchpad introduced masters as shareable objects that could be used in multiple locations in the scene [22]. Changing a property of the master would modify all objects that depend on this master. This was a way to reduce the number of actions required from the user when something must be changed.

*Properties* Presto is a document management system that enables users to tag documents with properties, e.g. *year=2012* [5]. Properties provide a uniform mechanism for managing, coding, searching, retrieving and interacting with documents. For example, users can define directories (i.e. a set) of documents using properties: either by extension (by putting elements into the directory), or by intension (with a query such as *size >500k*). Conversely to purely hierarchical structures, properties enable objects to be part of several overlapping sets.

*Graphical search* Graphical Search & Replace [13] allows users to search for elements based on their graphical properties (*designation*) and change at once a particular property for all found objects (*multiple scopes*). Applications like Illustrator provide such a tool but through a dialog box, not by direct manipulation.

*Surrogates* Surrogates are specialized interactors that allow users to interact with the surrogate instead of the domain object [12]. Similarly to classical inspectors, surrogates expose attributes that are common to objects, by automatically narrowing the surrogate to the lowest common ancestor. This enables users to interact with those values and modify several objects at once.

*User-defined macros and Programming by example* User-defined macros allow for automation of repetitive tasks [15]. The user proceeds with an example of the task to repeat, and an algorithm abstracts the actions, so as to enable application on other objects.

*Structuring for exploratory design* Some structuring techniques have been designed to support exploratory design. The list of reversible actions is an implicit mechanism to help users not to fear possible damages [23]. Side Views display previews of interactive commands [25]. Parallel Paths support alternative exploration by relying on an arborescence of creations instead of a linear history, and on the simultaneous views of parallel results (*comparison*) [26]. Acting on a node of the creation path enables users to manipulate the subsequent designs at once (*scope*).

### Structuring for designers

Interaction designers have already identified the need for many modifications with a low number of actions.

*Cognitive dimensions* In the cognitive dimensions of notation framework [8], the problem described in the software keyboard scenario is identified as "viscosity". It exhibits when the structure of the information contains a lot of dependencies between parts, which implies that a small change leads to numerous adjustments from the user. Viscosity is a hurdle to modification and exploratory design [9]. Since it may be costly to apply the changes, the user refrains from exploring alternatives. A solution to viscosity consists in creating an "abstraction", a "power command" that would act on several objects [9]. An abstraction is a class of entities, or a grouping of elements that users will handle as a single unit e.g. styles in a text document.

Abstraction can be costly. Learning, creating and modifying them require time and effort that should be balanced with investment in repeating a small sequence of actions to solve a small problem. Besides, abstractions can be a hurdle to exploratory design if they are required before any other

simple actions. Finally, abstraction may introduce hidden dependencies: some parts of the scene may depend on others in an invisible way, which makes it hard for the user to predict the effect of a change.

*Instrumental interaction and design principles* Direct [23] and instrumental [2] interaction techniques are efficient with a single object: they lower the number of required actions compared to other techniques, such as command lines, conversational dialogue, or modal interactions. Design principles related to instrumental interaction, such as reification (turning an object into a thing), polymorphism (applying the same change to different class of objects) and reuse (of past selection and interactions result) extend the scope of actions to multiple objects [2].

*Cost of interaction techniques* A particular technique is only better than another with respect to the task to accomplish: copy, modification, or problem solving (equivalent to exploratory design) [16]. CIS is a model that helps describe an interaction technique, analyze it, and predict its efficiency in the context of use [1]. CIS defines four properties for interaction techniques. Among them, Fusion is the ability of a technique to modify several work objects by defining multiple manipulations at once (*scope*), and Development corresponds to the ability offered to the user to create copies of tools with different attribute values.

### Structuring for programmers

The problems raised so far can also occur during development activities. For example, refactoring tools in IDEs is an answer to the need for multiple scopes of action: if the user changes the name of a method, the system applies this change on each call of the method, possibly in many classes or files. Styles can be implemented in a style language (e.g. CSS), with a hierarchical structuring. Changing a parameter in an intermediate node has an effect on its children. Tags in the Tk toolkit allow the programmer to structure objects in overlapping sets [21]. Changes can be applied to graphical shapes or to a tag, and thus to the set of objects that hold this tag (*scope*). Tags can be defined by extension (with *designated* objects) or by intension (with a predicate e.g. all blue objects) [21].

Prototype-based languages offer an alternative to class-based languages for object-oriented programming [14][20]. They offer a flexible creation model that allows sharing of properties and behaviors. Such mechanisms allow users to structure a hierarchy of prototypes and to act on several clones by manipulating a prototype in the delegation hierarchy. Morphic reifies prototypes and clones into graphic objects (called Morphs), and allows for their construction and edition with direct manipulation [18]. Tools have been designed to help structure a prototype hierarchy. For example, Guru is an algorithm that automatically creates a well-organized graph of prototypes, by factoring shared properties into new prototypes [19].

## REQUIREMENTS

In this section, we synthesize the requirements for the manipulation of objects through structures (Table 1). The synthesis is derived from the contextual inquiries we ran, and our analysis of the related work. Notably, the requirements are related to the set of tasks identified in [6] that are known to be difficult to perform with direct manipulation techniques. We have expanded and refined them in this section. We present 3 subsets of requirements: *managing sets of objects* (R1), *managing actions* (R2), *fostering exploratory design* (R3).

| | |
|---|---|
| Manage sets of objects (R1) | *Search* (R1.1) |
| | *Designate* (R1.2) |
| | *Modify* (R1.3) |
| | *Identify sets* (R1.4) |
| Manage actions (R2) | *Specify their nature* (R2.1) |
| | *Specify their parameters* (R2.2) |
| | *Specify the scope* (R2.3) |
| | *Perceive consequences* (R2.4) |
| Foster exploratory design (R3) | *Try* (R3.1) |
| | *Evaluate* (R3.2) |
| | *Short-term exploration* (R3.3) |
| | *Compare versions* (R3.4) |
| | *A posteriori structuring* (R3.5) |

**Table 1: Requirements**

Managing sets consists in *searching* (R1.1), and *designating* (R1.2) the objects that are part of a set. It is also necessary to *modify* (R1.3) the sets (add, remove elements). Finally, users must be able to *identify* (R1.4) the objects that belong to a particular set, or determine the sets a particular object belongs to.

Managing actions consists in *specifying their nature* (e.g. by clicking on an ‘alignment” icon, or a menu) (R2.1), *their parameters* (“vertical” or “horizontal”) (R2.2) and their *scope* (R2.3). *Perceiving their consequences* (R2.4) with appropriate feedback enables the user to realize the effects of its action after, and even before it is triggered [23].

In order to support exploratory design, it is important to provide users with tools that enable them to *try* (R3.1) and *evaluate* (R3.2) solutions during short-term exploration (R3.3), and *compare different versions* during middle-term exploration (R3.4) [24]. When satisfied with the results, users must be able to extend the modifications to other objects. If the system does not support this task efficiently, users will have to repeat the same actions to propagate changes (viscosity). Finally, if structuring is a solution to the viscosity problem, it is a hurdle to exploration if required a priori. Therefore, structuring should be made *a posteriori* (R3.5) i.e. when actions have already been done.

## INTERACTIVE TOOLS

We have explored a number of interaction techniques to offer new ways of interacting with multiple objects through structures. To design them, we involved the users we interviewed in a participatory design process, with 2 brainstorming and sketching sessions, and 5 evaluation sessions, as demonstrated in [17]. In the following, we cite the requirements that each feature is supposed to address. Requirements serve both as rationale to explain the design, and to help readers determine whether they are satisfied by our claims that the design fulfills the requirements.

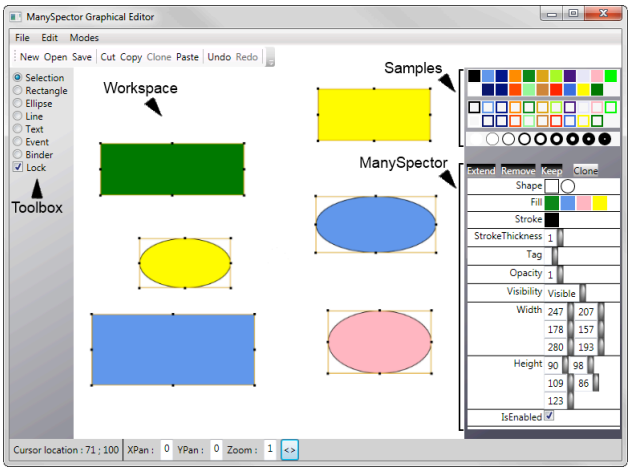



**Figure 4. Overview of the application. Center: workspace, top-right: samples; bottom right: inspector.**

### Overview

To illustrate the interactive tools, we have designed a graphical drawing application. There are four parts: a tool palette on the left side, a workspace in the middle, a sample panel on the top right corner, and an inspector on the bottom right corner (see Figure 4). The workspace is the main view, where users can create a new object by clicking and resizing. Selection is performed by clicking on an object or by drawing a rubber rectangle to encompass several items, as implemented in usual graphics editors. A bounding box with handles surrounds selected items.

The samples panel contains a set of values for shape (square, oval, T for text), fill color (represented by a colored square), stroke color (stroked-only colored square) and stroke thickness (stroked-only circle). In order to modify a property of an object in the main view, users can drag a sample and drop it onto the object. Feedback is shown as soon as the sample hovers over the object, in order for the user to understand the action and to assess the change before effectively applying it by releasing the mouse button. This enables the user to cancel the action, by releasing the button outside of any object (*R3.1 try, R3.2 evaluate, R3.3 short term, R3.4 compare, R2.4 perceiving consequences*). Drag and drop of samples also applies to a selection of objects. The interactions described so far are not entirely novel. The next sections present two tools with novel interactions.

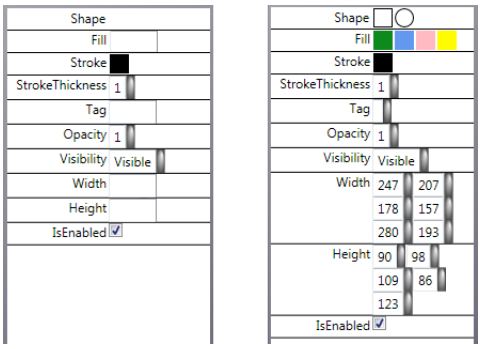



**Figure 5. The user's selection contains objects with varying shapes, fill colors, width, and height. A classical inspector (left) displays a blank fill for those properties, whereas ManySpector (at right) displays all different values.**

### Implicit structure: ManySpector, an enhanced inspector

An inspector (or property sheet [11]) is a window containing a vertical list of pairs of property name and value (e.g. shape: rectangle, color: green, thickness: 3). An inspector offers two services to the user: visualizing values with progressive disclosure and modifying them [11]. If multiple objects are selected, a classical inspector only displays values shared by all selected objects (e.g. stroke color in Figure 5, left). Users can change such a value, and the system reflects the change to all selected objects. The inspector does not display any value for properties for which there are multiples values (e.g. fill color in Figure 5, left). Users are thus not informed about those values, and sometimes cannot modify them through the inspector.

We have designed ManySpector, an inspector that displays all used values for a property given a set of differing objects. For example, in Figure 5-right, the Fill property displays all colors used by objects in the selection. Used values reveal an *implicit* structure of graphics, the sets of objects that share a value for a given property. Though not explicitly defined by the user, we think that such sets may be useful, since users sometimes think about objects with a graphical predicate ("all red objects"). We relied on the display of used values to design a set of interactions that offer new services for exploratory design and structure-based interaction: query and selection of objects with graphic examples, selection refinement, and properties modification on multiple objects.

The representation of a shared value in ManySpector actually reifies [3] both the value per se, and the set of selected objects that exhibits this property value. As a value per se, and similarly to the interaction with the sample panel, users can drag the shared value (considered as a value) from ManySpector onto (a selection of) objects in the main view to modify a property. If the shared value is numerical, users can hover over it and rotate the mouse wheel to increment or decrement it (*scope* and *specify actions*). Together with immediate feedback, this enables both exploration and precise adjustment of properties, thus reducing temporal offset [2] between action and feedback.

ManySpector limits the number of used values to half a dozen. If the number of used values is larger, a scrollbar enables the user to browse through all values. When the cursor hovers over a property placeholder, an animation enlarges it smoothly to reveal other used values.

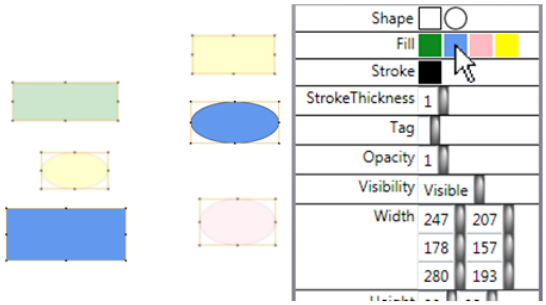



**Figure 6. The cursor is over the blue shared value of the fill property. Because they don't have this shared value, the green rectangle, the pink circle and the two yellow shapes are dim.**

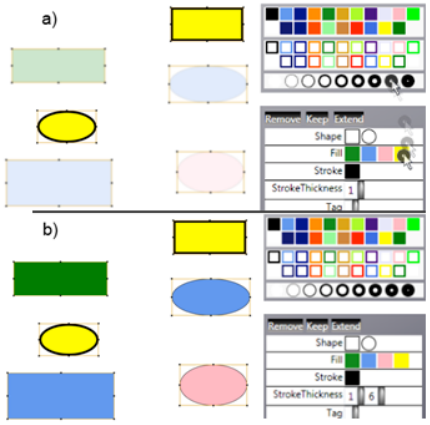



**Figure 7. Starting from Figure 5, a) the user drags a "stroke thickness: 6pt" sample over the "fill: yellow" shared value. Immediate feedback turns the stroke thickness of all yellow items to 6pt. b) the user has dropped the sample, the modification is applied.**

Since a shared value also reifies a set of objects, hovering over a shared value highlights the relevant objects while blurring others with a short animation (Figure 6). This makes it easy to figure out which set is made of what (*identify sets R1.4*), and to detect outliers and fix them. Users can drag a sample (a value) from the sample panel onto a shared value (considered as a set of objects) to modify at once a property for multiple objects (*R2.3 scope*) (Figure 7). Users can also drag a shared value (value) onto another shared value (set) (Figure 8).

To select objects, users can click on them in the workspace, or draw a selection rectangle. In order to refine the selection, users can use three meta-instruments (i.e. instruments that control instruments, here the selection): *Remover*, *Keeper* and *Extender*. The interaction consists in a drag and drop of the representation of the instrument onto a shared value. *Remover* throws out of the selection all objects that have this shared value (Figure 9). *Keeper* keeps in the selection the objects that have this shared value, and throws away the others. *Extender* adds to the selection all objects that are not selected but that possess this shared value. The instruments can also be dropped onto an object of the scene to add or remove it from the selection. These interactions extend the set of example-based queries introduced above (*R1.3 modify sets*).

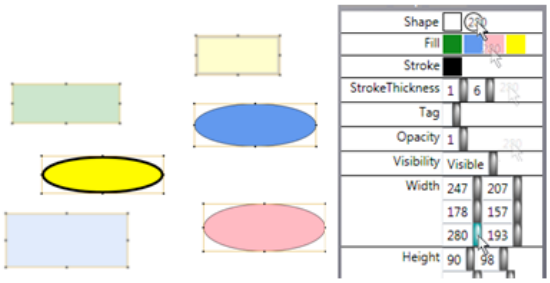



**Figure 8. The user drags the "width: 280" shared value and drops it on the "shape: circle" shared value. All circles in the selection now have a width set to 280.**

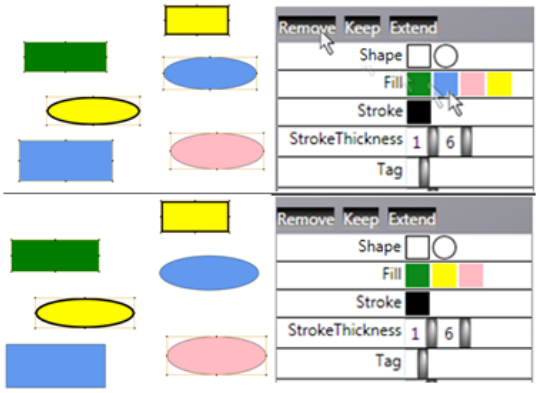



**Figure 9. The user drags the Remove tool onto the "fill: blue" shared value. Blue objects are removed from the selection.**

### Explicit structure: the property delegation graph

Besides ManySpector, we have explored an interactive tool that enables users to structure the content explicitly. Users can specify that a property of an object (the clone) depend on the property of another object (the prototype). A prototype is similar to a master in Sketchpad: when users change a property of a prototype by dropping a sample from ManySpector onto the prototype, all dependent clones are changed accordingly (*R1.3 modify sets, R2.3 scope*).

The interaction to specify a dependency is as follows (Figure 10): by clicking on an object, users can toggle the display of the properties around it. They can press on a property, draw an elastic link, and drop it onto another object as if they were dropping a sample. The clone object appearance reflects immediately the appearance of the clone for that property. Users can remove a link by pressing the mouse button in the blank space, drawing across the links to be deleted, and release the button.

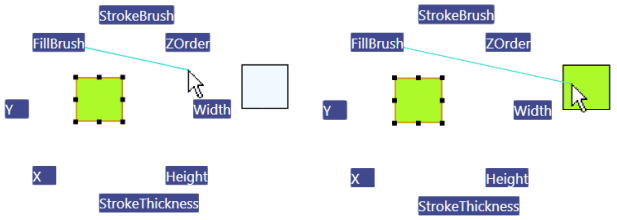



**Figure 10. The user draws a link between the fill property of the green object (the prototype) into the blue object (the clone) to specify a dependency. The fill color of the clone turns to the color of the prototype (green).**

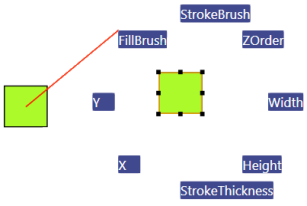



**Figure 11. The user has selected the clone to see the dependency.**

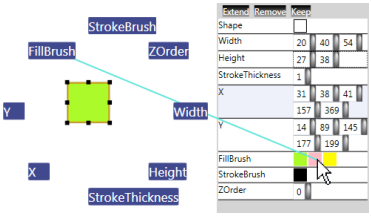



**Figure 12. The fill property is dragged onto a used value to specify that the fill property of a set of objects depend on the prototype.**

The system proposes two ways of creating new objects from existing ones: either by copying it or by cloning it (*R1.3 modify sets*). Copying is the regular copy operation: properties from the copy are independent from the properties of the source. Cloning enables users to get a clone, whose properties are entirely delegated to the copied object (the prototype) (Figure 11). By creating a clone, users minimize the number of actions required to specify a single difference with the prototype: if they copied instead of cloned, they would have to link all shared properties.

Explicit structuring is supposed to bring more action power, at the expense of increasing viscosity and hindering exploratory design since users have to manage a structure. We have lowered these drawbacks with a posteriori structuring and by leveraging off ManySpector. For example, choosing to clone or to copy may be premature at the moment of the creation of a new object from an existing one. To solve this problem, users can decide to change them to a copy or a clone after the creation of the object (*R1.3 modify sets, R3.5 a posteriori structuring*). This is made possible by tracing the history of objects, and how they were created. Toggling between copy and clone only affects the properties that were not set explicitly by the user. Another problem is to interact with similar objects in order to make them depend on a prototype. A viscous solution would be to interact with each object and making it a clone of the prototype. A more efficient solution consists in selecting the objects that are to be clones, and in dropping the property of the prototype onto an object of the selection (*R1.3 modify sets, R3.5 a posteriori structuring*). Users can also drop the property onto a shared value in ManySpector (Figure 12), which links all objects sharing that value to the prototype.

The property delegation graph is an extension of the delegation tree found in prototype-based languages [14]. However, with a tree, objects cannot have multiple parents. For example, the scene tree available in illustrator may be helpful to conceptualize the scene, but is unable to help specify cross-branches relationships. Conversely to a tree, a node in our graph of properties can have multiple parents. This enables users to be more specific about the parent that holds a particular property: a node can delegate 'fill' to a prototype A, and 'stroke-width' to a prototype B.

### Discussion about the design

The interactions are consistent: they all use modeless interaction based on drag and drop, be it from or on an object on the scene, a shared value, or a prototype. With immediate feedback and a posteriori structuring, they also support exploratory design. The properties are immediately visible (no need to devise a query): users can try and test by hovering over and off the used values, and assess the results thanks to immediate feedback without applying the change (button still pressed).

The interactions we devised can be considered as a kind of surrogates [12]. We have expanded them by explicitly taking into account the interaction to manage the selection and explicit structuring. Furthermore, our version exposes not only common properties but also all used values, which makes direct the access to more subsets and expands notably the scope of interactions. Of course, existing systems enable users to obtain the same final results, and even by relying on similar concepts (flash, sketchpad). Those systems actually provide the same *functionalities*, but not the same *interactions*. For example, existing tools do enable users to perform a graphical search, but with an indirect manipulation (through a menu and a dialog box). This prevents users from quickly trying and testing changes and hinders exploratory design. In addition, interactions are not well integrated *e.g.* in Illustrator, there is a tree view, but users can use it only to select a branch then apply a limited set of changes on the selection.

As such, the prototypes have issues. For example, more work needs to be done with respect to scalability: ManySpector is not able to handle very large sets of used values. The solution with a scrollbar and progressive disclosure may not be sufficient. The prototype/clone view also needs more work: if the links are numerous, the scene may result in a mess of tangled links. Again, progressive disclosure is a possible solution but we are also exploring other representations and interactions [10]. Furthermore, the system does not check for cycle when the user tries to link two properties. Appropriate feedback is necessary to prevent it, such as displaying the links to show a potential cycle when hovering over a property.

## USER STUDY

We have argued in the previous sections that our tools are novel, consistent and effective for performing structure-based interaction. Assessing those claims is not a straightforward task. We were especially concerned with the understandability of the used values concept, and the fact that they refer either to a value or the set of objects that share this value. Would it be too difficult for users to grasp the shared value concept and linked properties? Even if users understand them, how would they struggle when trying to use them to interact with multiple objects? Finally, can users translate high-level problems into graphical interactions with used values and linked properties?

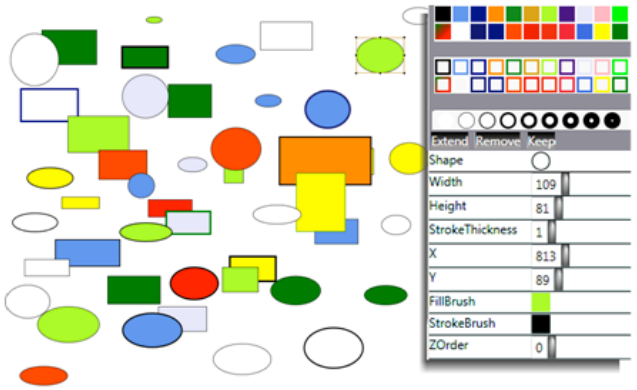



**Figure 13. The scene containing many objects.**

### Tasks

The evaluation session was divided into three parts, each dedicated to one of the three questions above. The first part was devoted to a tutorial that teaches users about used values and links, and how to interact with them in the graphical editor. The two other parts are scenarios that were designed so that they implement the requirements.

In the tutorial, we instructed users to create a few objects, link them, change their color or stroke thickness, with a single object or a set of objects. The tutorial lasted 10min and included 15 simple tasks. Users were actually manipulating the mouse and performed interactions while they were listening to our instructions. The goal of this tutorial was not only to instruct users, but also to see if they understood the design. We assessed their understanding by observing them perform small tasks with no instructions and by asking them if they were confident in their understanding. We did not assess discoverability since we began with a tutorial. This aspect is left for future work.

The second part of the session was an actual test. The test was still using the graphical editor, but this time with a scene containing multiple (50) differing objects (see Figure 13). We asked users to perform more complex tasks such as 'change the thickness of all yellow circles to the maximum of all thicknesses'. We did not give any instructions, and left users perform the tasks by themselves. One of the expected benefits of used values is to help users select a set of objects with minimal interactions. Hence, we designed the tasks to make traditional selection (i.e. a selection rectangle, or adding shapes to the selection by shift-clicking on them) more and more difficult either because they involve multiple objects (*scope R2.3*), or because they involve graphical properties that are not perceptually pre-attentive (*search R1.1, identify sets R1.4*). For examples, the task "change all circles' color" is difficult because users need to find all circles in a scene, a visual task known to be non pre-attentive and that requires a cumbersome one-by-one scan of graphical objects (try on Figure 13). Users were free to carry out the tasks the way they want, either by selecting shapes with the traditional way or using ManySpector (*designate R1.2*). The goal of this second part was to assess the extent to which users would rely voluntarily on used values and links, whether they would be able to perform non-trivial graphical tasks (*specify action R2.1 and parameters) R2.2*), and how well they could interact with used values and links.

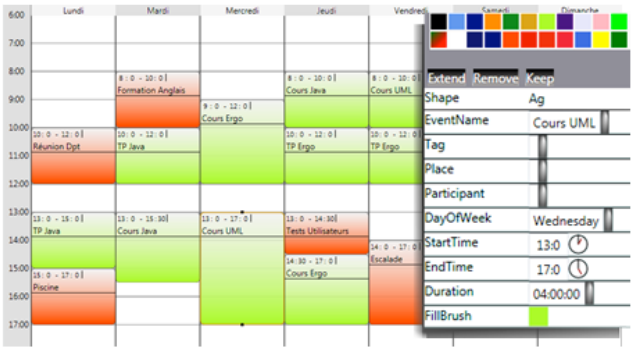



**Figure 14. The calendar view.**

The third part involved a calendar application. Users were manipulating events on a week view (see Figure 14). Events are represented with rectangles with a title text and a start hour text. They are placed horizontally according to day of occurrence in the week and vertically according to the time in the day. The screen is filled with seven columns, one per day in the week. Instead of graphical properties, the ManySpector window contained calendar-related properties such as start, duration, title etc. as in the iCal inspector. Conversely to iCal, ManySpector displays used values. This allows for modification of unrelated events, while iCal allows for modification of multiple repeated (i.e. recurring) events only. We provided a partially filled schedule and we asked users to act as if they were teachers trying to schedule lecture sessions during the week with a schedule "manager" (the role we played). For example, we asked them to place a 2-hour long lecture Wednesday afternoon. Then we told them that when we said "place a lecture at 10am", we actually meant "10:15am", so they had to change all "10am" lecture events to "10:15am" (*a posteriori structuring R3.5*). The goal of this third part was to assess whether users could translate higher-level tasks to graphical interactions with our tools. The tasks were high-level, and required users to *try R3.1*, *perceive the consequences R2.4*, *evaluate R3.2* and *perform short-term exploration R3.3*.

Since the calendar scene contained few elements only (~15), we were expecting that users would rely on traditional selection. Hence we asked them to use ManySpector instead of the traditional selection.

### Subject profiles

We performed the tests with five subjects. Three of them use calendar application in a day-to-day basis, one of them was a graphical designer used to applications such as Illustrator, and one was a casual user of graphical tools such as presentation software. They were all aware about the viscosity problem that might occur when using such tools. Only the graphical designer was involved in the participatory design process, hence four users discovered the interactions for the first time.

### Procedure

We asked subjects to think aloud [7] while they were acting. We observed them and logged what they tried, whether they struggled, made errors or succeeded. At the end of the second and third part, we made them fill a questionnaire to rate the difficulty and cumbersomeness of the tasks, and the usefulness of the design with a Likert scale from 1 (negative) to 5 (positive). Results are given in the following, with the mean and the standard deviation.

### Results

We did not notice serious understandability problems. Users were able to manipulate shared properties and links, and succeeded in performing simple tasks at the end of the tutorial. When asked about their confidence, some of them felt that they needed some learning "*to do it well*". We showed them many interactions, but even if the interactions are well integrated, users felt that they could not get familiar with them within such a short time. In addition, because there were several possibilities to accomplish tasks, users were always eager to find the best way of accomplishing it, which adds to their feelings. Our confidence into users' understandability got stronger when we witnessed that they got more capable as they were performing the second and third part. We even observed users trying interactions that we did not designed but that were perfectly meaningful, such as using selection instruments (keep, remove) directly on samples to avoid the necessity to perform a selection of the entire scene, dropping a value onto a property name to apply it to all objects, or dragging a sample next to existing used values to extend the selection. This suggests that the design was consistent and predictable.

We did notice some difficulties when users performed more complex graphical tasks in the second part (*ease of translation in graphic scenario*: mean: 3.6, stddev: 0.5). This can be explained by the fact that users were still learning the interaction. They also told us that the tasks were rather abstract. In fact, since the tasks were purposely complex, they lacked significance (none performed 'change the thickness of all yellow circles to the maximum of all thicknesses' in real-life). They struggled to understand and memorize them, which hindered their ability to devise a solution. The four non-graphical designers found the requests much less difficult in the last part with the calendar application and meaningful tasks. Still, all subjects were able to accomplish every tasks of the second part by themselves. (*mean of the easiness of the 9 subtasks of the graphic scenario*: 4.6; 0.5).

We were wondering about voluntary use. We observed what we expected: with tasks that involve pre-attentive properties (such as color-oriented one: 'turn yellow objects into red'), subjects were sometimes still using a traditional selection. However, they turned by themselves to used values with non-pre-attentive tasks, or when the number of objects was too important. They also used links when we asked them to repeat an interaction on the same set of objects: after a number of repetitions, some subjects turned a specific object into a master. This enabled them to be more efficient than devising a selection again with the ManySpector. All kinds of interaction were performed (with samples, used values, links), and all combinations of source and destination for drag and drop were witnessed.

We did not notice difficulties when users had to translate higher-level tasks into interactions in the calendar test (*ease of translation in calendar scenario*: 4.2; 0.8). We witnessed a tendency to use traditional selection for very simple tasks. When we forced users to employ our interactions instead, they did not have difficulties to do so (*mean of the easiness of the 7 subtasks of the calendar scenario*: 4.7; 0.5). This suggests that the interactions can be applied to other contexts than graphical edition.

Even if we did not plan to evaluate usability, the tests revealed some issues such as the difficulty of interacting with the text boxes. Users also found limits to the interactions we proposed: in some cases, users would have liked to keep objects based on a combination of values instead of a single one. As expected, links lacked visibility and legibility when numerous.

All in all, the study allowed us to answer positively to our concerns: the tools fulfill the requirements since users were able to understand the interactions, could perform complex graphical tasks with them and could translate higher-level tasks into them. Users judged ManySpector very useful (*ManySpector usefulness*: 4.8; 0.4). They liked explicit structuring with links though not as much as used values (*links usefulness*: 4.4; 0.9). They also praised the fact that there was no imposed strategy and that they could perform tasks their way.

## CONCLUSION

We have tackled the problem of interaction with structures, and interaction with content through structures. We have defined a set of requirements and have explored a set of consistent interactions that provide partial answers to the

requirements: ManySpector, an inspector for multiple objects, and explicit delegation links. A study showed that users are able to perform complex graphical tasks with them. The examples involved a drawing editor and a calendar but the requirements and interactions are not specific to these applications, and can be applied to others.

Our interactions suffer from some problems such as scalability (though this may not be a problem for e.g. the calendar) and legibility. Other designs are possible: we are currently investigating other forms of explicit structuring with no links. We also plan to assess how well those interactions support exploratory design.


## ACKNOWLEDGMENTS

We thank all participants of the workshops, and our colleagues for early and late feedback on the paper.


## REFERENCES


1. Appert, C, Beaudoin-Lafon, M, Mackay, W. E. Context matters: Evaluating Interaction Techniques with the CIS Model. *Proc. HCI'04*, 279-295. Springer Verlag, 2004.
2. Beaudouin-Lafon, M. Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. In *Proc. CHI 2000*, ACM, 446-453.
3. Beaudouin-Lafon, M. and Mackay,AW. E. Reification, polymorphism and reuse: three principles for designing visual interfaces. In *Proc. of ACM AVI 2000*, 102-109.
4. Conversy, S., Chatty, S. and Hurter, C. Visual scanning as a reference framework for interactive representation design. In *Information Visualization*, Sage, 2011.
5. Dourish, P., Edwards W.K., LaMarca, A. and Salisbury, M. 1999. Presto: an experimental architecture for fluid interactive document spaces. *ACM Trans. Comput.-Hum. Interact.* 6, 2 (June 1999), 133-161.
6. D. M. Frohlich. The history and future of direct manipulation. *Behaviour & Information Technology*, 12(6): 315–329, 1993.
7. Ericsson, K., & Simon, H. (May 1980). Verbal reports as data. *Psychological Review*, 87 (3): 215–251.
8. Green, T.R.G., Cognitive dimensions of notations, *People & Computers V*, 1989, Cambridge Univ. Press, 443-460.
9. Green, T.R.G, and Blackwell, A. Cognitive dimensions of information artifacts: a tutorial. (Version 1.2), 1998.
10. Holten, D., Isenberg, P., van Wijk, J. J., Fekete, J.-D. 2011, An extended evaluation of the readability of tapered, animated, and textured directed-edge representations in node-link graphs, *IEEE PacificVis*, 195–202.
11. Johnson J.A., Roberts T.L., Verplank W., Smith D.C., Irby C.H., Beard M., and Mackey K. The Xerox Star: A retrospective. *IEEE Computer*, 22(9): 11–29, 1989.
12. Kwon, B., Javed, W., Elmqvist, N., and Yi, J.-S. Direct Manipulation Through Surrogate Objects. In *Proc. of ACM CHI 2011*, 627-636.
13. Kurlander, D, Bier, E.A. Graphical Search and Replace. In *Proc. of ACM SIGGRAPH '88*, 113-120.
14. Lieberman, H. 1986. Using prototypical objects to implement shared behavior in object-oriented systems. In *Proc. of OOPLSA '86*. ACM, 214-223.
15. Lieberman, H. Your Wish is my command: Programming by example. Morgan Kaufmann, 2001.
16. Mackay, W.E. Which interaction technique works when?: floating palettes, marking menus and tool-glasses support different task strategies. In *Proc. of AVI '02*. ACM, 203-208.
17. Mackay, W.E. Using Video to Support Interaction Design. DVD Tutorial, *CHI'02*, ACM.
18. Maloney, J.H. and Smith, R.B. 1995. Directness and liveness in the morphic user interface construction environment. In *Proc. UIST '95*. ACM, 21-28.
19. Moore, I. 1996. Automatic inheritance hierarchy re-structuring and method refactoring. In *Proc. of OOPSLA'96*. ACM, 235-250.
20. Myers, B. A., Giuse, D. A. and Zanden, B V. Declarative programming in a prototype-instance system: object-oriented programming without writing methods. *SIGPLAN Notice* 27, 10 (1992), 184-200.
21. Ousterhout, J. K. Tcl & Tk Toolkit. Addison-Wesley, 1994.
22. Sutherland I.E. 1963. Sketchpad: a man-machine graphical communication system. In *Proc. of AFIPS'63*. ACM, 329-346.
23. Shneiderman, B. Direct manipulation: a step beyond programming languages. *IEEE Computer* 16(8), 57–69, 1983.
24. Terry, M. and Mynatt E.D. Recognizing creative needs in user interface design. *Proc. of Creativity & Cognition*. ACM, 38-44, 2002.
25. Terry, M. and Mynatt, E. D. Side views: persistent, on-demand previews for open-ended tasks. In *Proc. of UIST 2002*.ACM, pp. 71-80.
26. Terry M, Mynatt E.D, Nakakoji K, and Yamamoto Y. 2004. Variation in element and action: supporting simultaneous development of alternative solutions. In *Proc. of CHI '04*. ACM, 711-718.
27. Ungar, D, Smith R, B. SELF: The Power of Simplicity. In *Proc. of OOPSLA '87*. ACM, 227-242.

# Flights in my Hands: Coherence Concerns in Designing Strip'TIC, a Tangible Space for Air Traffic Controllers

**Catherine Letondal**[1], **Christophe Hurter**[1,3], **Rémi Lesbordes**[2], **Jean-Luc Vinot**[1,3], **Stéphane Conversy**[1,3]

ENAC
7 avenue Edouard Belin
31055 Toulouse, France

DSNA/DTI/R&D
7 avenue Edouard Belin
31055 Toulouse, France

IRIT, Université de Toulouse
118 route de Narbonne
Toulouse Cedex 9, France

{catherine.letondal, christophe.hurter, jean-luc.vinot, stephane.conversy}@enac.fr
remi.lesbordes@aviation-civile.gouv.fr

## ABSTRACT

We reflect upon the design of a paper-based tangible interactive space to support air traffic control. We have observed, studied, prototyped and discussed with controllers a new mixed interaction system based on Anoto, video projection, and tracking. Starting from the understanding of the benefits of tangible paper strips, our goal is to study how mixed physical and virtual augmented data can support the controllers' mental work. The context of the activity led us to depart from models that are proposed in tangible interfaces research where coherence is based on how physical objects are representative of virtual objects. We propose a new account of coherence in a mixed interaction system that integrates externalization mechanisms. We found that physical objects play two roles: they act both as representation of mental objects and as tangible artifacts for interacting with augmented features. We observed that virtual objects represent physical ones, and not the reverse, and, being virtual representations of physical objects, should seamlessly converge with the cognitive role of the physical object. Finally, we show how coherence is achieved by providing a seamless interactive space.

**Author Keywords:** Tangible interaction; Augmented paper; Pen-based UIs; Distributed cognition; Participatory design; Ethnography; Air Traffic Control; Transport; Security.

**ACM Classification Keywords:** H.5.1. Information interfaces and presentation (e.g., HCI): Miscellaneous.

**General Terms:** Human Factors; Design; Measurement.

## INTRODUCTION

Mixed interaction design has been studied through several approaches: tangible user interfaces (TUIs) [35, 11], augmented reality (in the sense of [26]), and all approaches that can be described as reality-based interaction [15]. Several models [35, 11], taxonomies [7], frameworks [12, 35, 15, 21] or guidelines [17, 34] have been published that inform the design of mixed interactions. They address its complexity, mainly through issues related to coherence of the mapping or coupling of physical and virtual elements [35].

However, recent literature points out the limits of TUIs. In TUI litterature (e.g [21]), coherence is achieved through mapping, i.e when the physical and the digital artefacts are "seen" as one common object. Mapping-based coherence thus involves how representative a physical object is of a virtual one. This has been challenged [6]. First, the claim that TUIs enable to physically manipulate abstract data has been questioned [20]. Second, addressing mixed interaction complexity cannot rely solely on a mapping-based coherence. In [26], Mackay warns about not spoiling the understanding that users have about the laws that dictate the behavior of physical objects by a behavior that is dictated by the humans who build the virtual system. In [13] Hornecker further questions the assumption that affordances of the physical world can be seamlessly transferred to computer-augmented situations: users actions are not always predictible, nor do their expectations about the behavior of the system, and it is not obvious for the designer to know which prior knowledge of the real world will be invoked.

We profited from the redesign of an Air Traffic Control (ATC) environment, an operational, complex system already based on basic mixed interaction, to gather new knowledge on mixed interaction design. The system (named Strip'TIC [14]) explores a solution that integrates interactive paper, handwritten notes and digital data, using digital pen and augmented reality technologies.

This paper presents the results of this investigation. Notably, we present how we addressed mixed interaction complexity through a view of coherence that departs from mainstream TUI models. In our context, physical objects and associated manipulations have an inner coherence due to their cognitive role as external representations [19], that designers must respect. A consequence is that physical objects represent mental objects rather than virtual ones. Furthermore, the virtual objects actually represent the physical ones, and not the reverse, which brings constraints on their design. Finally, we show how coherence is achieved by providing a seamless interactive space.


*CHI 2013*, April 27–May 2, 2013, Paris, France.

## RELATED WORK

Our work relates to interface approaches that employ physical means of interaction or reality based interaction [15]. In this field, following the tradition of metaphor-based usability, a first research goal has been to find how to design interfaces that mimic the real world: the main idea was to build on prior knowledge to foster usability [15]. Research work following this goal includes reflection on how to give a physical form to digital objects [35], and also on mapping and coupling physical objects and their digital counterpart to enhance the coherence of the metaphorical relationship. An important issue has been to define tangible interfaces, either in terms of interaction models, e.g. [35], or through frameworks that describe the type and qualities of various dimensions, such as levels of mapping, metaphorical dimension [7] or concepts of containers, tokens and tools [11]. Characteristics of couplings are also described: in [7], the embodiment dimension describes the level of integration of input and output in tangible systems, while [21] classifies coupling according to the degree of coherence, as measured through several properties of links between virtual and physical objects.

A significant part of tangible interface research aims to explain not only prior knowledge or metaphorical scales, but also properties and affordances of the physical world these interfaces rely on [8, 12, 20]. In [12], Hornecker et al propose an analysis of physical interactions through four perspectives for the design of tangible interfaces: tangible manipulation, spatial interaction, embodied facilitation and expressive representation. Affordances of paper have been studied in depth by [32]. In [25], Mackay analyzes more specifically how paper strips support ATC activity. In [34], Terrenghi et al analyze tangible affordances through comparisons of different kinds of interactional experiences performed by similar physical and digital systems, such as comparing multiple items or creating spatial structures, with the aim of designing better digital systems.

Another field of research, instead of metaphorically extending digital systems to the real-world, aims at extending real-world objects by linking them to digital features through augmented reality [26]. This field notably includes augmented paper or paper computing research [33] which aims at integrating paper and digital documents.

Several views unify tangible interfaces and augmented reality, such as reality based interaction [15]. For designers, a more general issue thus becomes the allocation problem to either virtual vs physical objects. [17] for instance discusses how physical or virtual objects enable various tasks, such as disambiguating objects, supporting eyes-free control or avoiding mode errors. Some authors compare digital and physical according to efficiency in relation to specific tasks. In [28], McGee et al evaluate whether a paper-augmented system performs as efficiently as a digital system without losing the positive properties of paper, but rather focuses on the cost of recognition errors than on allocation issues.

Other approaches broaden TUIs definition. In [12], Hornecker et al rethink TUIs as interactive spaces, focusing on the quality of the corresponding user experience. In [6], tangibility is proposed as a resource for action instead as just an alternative data representation.

## ATC ACTIVITY AND ITS INSTRUMENTATION

As said above, our study on mixed interaction is grounded on air traffic control (ATC). Air traffic controllers maintain a safe distance between aircraft and optimize traffic fluidity, usually working by pair. The planner controller predicts potential conflicts between aircraft. The tactical controller devises solutions to the conflicts, implements them by giving instructions to the pilots by radio, and monitors the traffic. Air traffic controllers currently use a combination of computer-based visualization (e.g. radar image) and tangible artifacts (paper strips) to manage traffic [25].

### Specific control situation

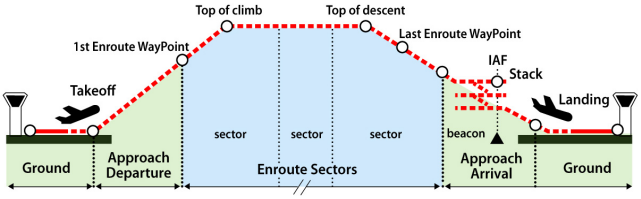



**Figure 1: phases of flight and associated control areas. Final approach control manages aircraft from the stack and the IAF (Indicated Approach Fix) point before landing.**

We have decided to ground our study on a specific ATC situation, the Approach Control (Figure 1). The traffic of Approach is complex and dynamic: aircraft are most likely ascending or descending; minimum separation in time and space between aircraft goes from 5Nm (8km), through 3Nm in the Approach area; to 2.5Nm just before landing to ensure 2 minutes between two consecutive planes. Therefore the time of analysis and action decreases rapidly as the aircraft get closer to the field.

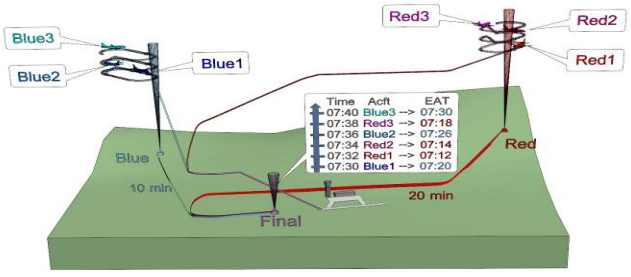



**Figure 2: a stack and calculation of the Estimated Approach Time (EAT).**

For traffic optimization purposes on a busy airfield, controllers try to get as close as possible to the maximum runway capacity i.e. the number of takeoffs or landings per hour (Figure 2). This is challenging since they need to optimize aircraft ordering and separation with a mix of takeoffs and landings. When the runway capacity is exceeded, controllers can delay the arrival of aircraft by piling them up into a ‘stack’ and making them perform horizontal loops. The first aircraft in the stack is assigned to

the lowest altitude. Each new aircraft entering the stack is piled upon the previous one. The first aircraft that leaves the stack is the lowest one. When an aircraft leaves the stack, the controllers order each remaining aircraft in the stack to descend to the next lower altitude.

The management of the Arrival sector can even be split between two controllers: a controller for the management of incoming aircraft and the management of the stacks, and a controller guiding and sequencing aircraft from the stacks to the final axis of the runway. Splitting (or degroupment) is a critical phase since controllers have to reallocate a set of strips on an additional stripboard. Paper strips can either be physically passed between controllers or be duplicated. In this case, the last information handwritten on the strips must be reported orally to the supporting controllers.

### Current instrumentation and tentative improvements

Controllers do not feed the system with the instructions they give to the pilot, neither through the computer-based part of the system, and of course nor through the paper-based part. This prevents the potential use of automation to help controllers regulate the traffic more efficiently and in a safer way. However, controllers do hand write the instructions on the strips to remember them. This has led the Airspace authorities in the EU and the USA to replace paper with digital devices (dubbed "electronic stripping" or stripless environment) in the hope that the instructions could be fed to the system. Although electronic stripping has been constantly improving during recent years, there is still reluctance to its being adopted. We suspect that such reluctance is partly due to the fact that screens do not offer the same level of interactivity as paper. In fact, the designers of electronic systems have devoted considerable effort to replicate interactions on the paper, be they prospective [29, 2], or operational (Frequentis SmartStrips or NAVCANStrips).

Considering the previous remarks, mixed interaction may be an appropriate approach to the improvement of the ATC environment. Caméléon early project [24] explored various technological alternatives to electronic stripping: transparent strip holders whose position could be tracked on a touch-screen, a pen-based tablet with no screen but with regular paper. However, these early prototypes were built with the technology of the mid-nineties and not all possibilities could be explored, especially those based on augmented paper.

### Significance for research on mixed interaction

ATC has a number of properties that may shed a new light on mixed interaction. ATC is real: the realness of the activity can act as a magnifier of aspects about mixed interaction that would be overlooked with artificial activities. ATC is life-critical: even if accidents are rare (because the way the system works helps prevent problematic situations) some circumstances can lead to life and deaths situations. Controllers involved in the design of tools and procedures are constantly aware of their responsibilities, which make them cautious and concerned. ATC is time-constrained: since a flight cannot stop in the sky, orders must be given under constraints. This makes time-performance an important concern. ATC is heavily designed for performance: tools and procedures have been refined and tuned for years by their own users, which has led to a high level of safety and capacity. This stresses the usefulness and usability of new proposed features. ATC is demanding in terms of human cognitive capability, and qualification on a complex area can take years. Even subtle aspects of the instrumentation may have an impact on cognitive load and deteriorate performance. ATC controllers are extremely concerned by the instrumentation of their activity. They reflect on the tools and their procedures and are eager to improve them based on a deep internal knowledge. Even if all aspects are not new (e.g. Reactable [16] for real-time and reality), we were expecting that the combination of those properties would raise the level of implication, reality, deepness and details during the discussions.

## PROTOTYPE DESCRIPTION

We have designed Strip'TIC, a novel system for ATC that mixes augmented paper and digital pens, vision-based tracking and augmented rear and front projection [14]. The paper strips, the strip board and the radar screen are all covered with Anoto Digital Pen patterns (DP-patterns). DP-patterns are small patterns (< 1mm width) used by the pen's infrared digital camera to compute the location of the pen on the paper.

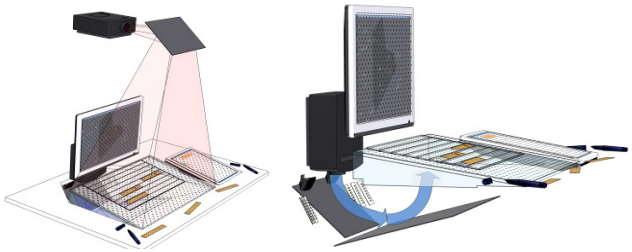


**Figure 3: Strip'TIC. Top projector, radar screen, strip board, and side screen (left). Bottom projector casting images on a semi-opaque strip board, two mirrors, infrared LEDs, and webcams (right). Black dots depict pen-sensitive areas.**

Users' actions with the digital pen are sent in real-time to IT systems wherever they write or point. Users can draw marks (e.g draw information on paper strips), write text (e.g. write aircraft headings), and point out objects (e.g. point out aircraft on the radar screen). The stripboard itself is semi-opaque: this enables bottom projection on the stripboard and strip tracking thanks to AR patterns printed on the back of the strips (Figure 4). Another projector displays graphics on top of the stripboard and on top of the strips. (Figure 3). Controllers can manipulate paper strips as they are used to with the regular system.

A cornerstone aspect of Strip'TIC is the mix between real and virtual strips. When a paper strip is put down onto the strip board, the tracking system recognizes it and projects a virtual strip under the paper strip with the bottom projector. Virtual strips are slightly larger than paper strips, which makes the paper strip borders 'glow' and acts as a feedback

for the recognition of the strip. When lifting up a paper strip, controllers can use the digital pen to interact with its corresponding virtual strip. They can move it by performing a drag on its border, and also write on it. When setting the paper strip down onto the board, the virtual strip is aligned under it. The then-occluded handwritten notes of the virtual strip are projected on the paper strip (Figure 6).

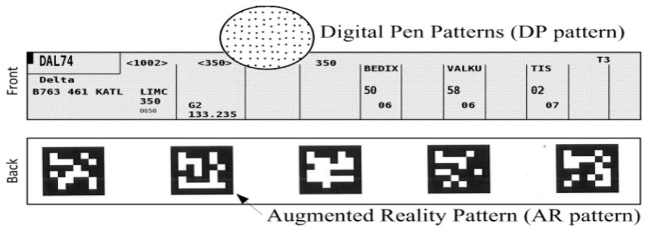



**Figure 4: Front and rear of paper strip. Each strip corresponds to a flight, and displays information such as the level of entry, the route and a timed sequence of beacons the flight is supposed to overfly while crossing the sector.**

We have implemented numerous features that rely on the combination of those devices: highlighting aircraft on radar when pointing on a real or virtual strip and vice-versa, projecting solid colored rectangles to colorize paper strips, adding real-time information on the strips (Figure 5), display of recognized hand-written texts, or even expanding paper strips with a virtual extension to address strip fixed size. Though technologically complex, the system is working in real-time and reactive. When introduced to Strip'TIC, controllers are usually eager to use it and discover all its features. The features that we specifically explored are discussed in the next sections.

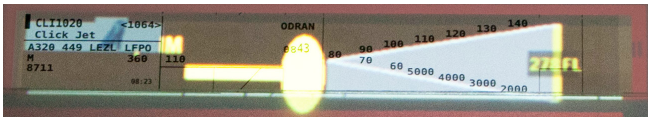



**Figure 5: Paper strip with projected information. Highlighted beacon, distance aircraft-beacon, aircraft name, aircraft vertical profile and current position, current altitude.**

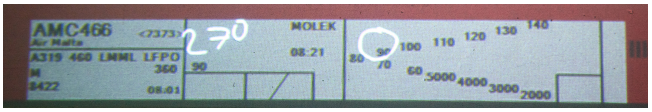



**Figure 6: Paper strip with projected handwritten information.**

## METHODOLOGY

Our study was conducted as design-oriented research [5]. Hence the goal of our work was not to produce a 'final' prototype, but rather to gather new insights about mixed interaction through grounded discussions with users about the support of their activity by the prototype.

### Designers: users and researchers

The observation sessions involved nine experienced controllers (more than 5 years experience) and six apprentices, and the design and walkthrough sessions involved nine other experienced controllers. All were French controllers involved in the three types of control (en route, approach and tower) from Bordeaux, Orly, Roissy and Toulouse. Our team is composed of 4 HCI designers and researchers (visualization, tangible and paper-based interaction, graphic design) and a controller.

### Sessions with users

We conducted a series of iterative user studies, ranging from field observations and interviews, both transcribed and encoded, participatory design workshops involving video-prototyping, and design walkthrough sessions where the controllers tried the prototype by running scenarios such as ungrouping, conflict detection, stack management, etc. During these studies, we observed controllers and discussed with them to understand important aspects of their activity related to our design. We also experienced technical solutions with the Strip'TIC prototype, either to test it or to demo it and get immediate feedback. We let controllers play with it and react by proposing ideas, in order to get insights on how to co-evolve the activity with them toward a mixed system. In total, we completed 4 observation sessions in real context, 3 observation sessions of training controllers, 4 interview sessions, 5 brainstormings and prototyping sessions and 4 design walkthrough sessions. We also demoed our prototype to 21 French controllers, collecting 13 filled-in questionnaires, and to controllers from Germany, Norway and UK to get informal feedback.

### Study conduct

Between the sessions with users, we implemented the ideas raised during the sessions. The fact that a feature would 'improve' the course of the activity was not the sole reason for further investigation. Instead, we were attentive to users' reactions and focus, especially when they were discussing the status of the artifacts, or if they considered them different based on some aspects (e.g. virtual or real). We were also attentive to users' discussions that may spark novel findings on mixed interaction and develop prototypes to investigate the raised issues further.

About 30 augmented features have been explored and prototyped during two years, using Wizard of Oz, paper, paper + video, Flash/Flex, or PowerPoint, and many have been implemented. In addition to the features described in Prototype section, we explored physical/virtual objects lifecycle management involving virtual but also physical strip creation (print/re-print), various interactions with physical strips (gestures, oral, pen-based) and application domain features such strip grouping, conflict or charge detection, various computation to support actions and decisions (distances, dates...), transitory state management, strip extensions, stripboard structure management (stacks, runways, simulation), temporal information (timelines, timers) (Figure 12), various informational features, macro commands, communication between controllers, and drawing or annotation on the screen.

## DESIGNING MIXED INTERACTION IN ATC

We present in this section our reflections and observations gathered during the design of Strip'TIC. The first part clarifies basic allocation principles in the case of ATC, often already reported in the literature, that we review and

supplement in the first part of this section. The second part addresses complexity and introduces coherence issues. It shows that they relate more to mixed physical/digital behavior than to mapping. The third part analyzes ATC temporal processes to introduce externalization as an important pattern to mitigate allocation problems and to design consistent tangible interactions.

### Physical/virtual allocation principles

*Positive properties*

Paper strips exemplify several aspects of physical interaction, as described in [12]. Tactile properties make the strips graspable and enable lightweight interaction, such as slightly shifting a strip to trigger attention. Non-fragmented visibility of the stripboard and performative properties of gestures enable awareness among controllers. Spatial affordances support reconfiguration, and physical constraints with the stripboard format make some interactions easier or more difficult. In our study, observed affordances of physical strips are also in line with findings from [25], either regarding how manipulations of the strips helps build the picture of the current situation, or how the physical setting supports subtle cooperation and mutual awareness, for instance enabling non-intrusive cross-checking by working independently on different problems within the same collection of strips. Mackay also shows how bimanuality enables efficient handling of complex problems and why flexibility of paper [32] and handwriting may support rapid adjustment to evolving rules.

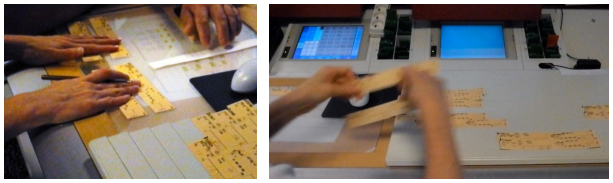


**Figure 7: a) one stack in each hand; b) bi-stack bi-manual strip transfer from the planner board to the tactical one.**

In our study, we focused on approach context, which involve many physical manipulations. We observed for instance how physical constraints were leveraged to help automate decisions or prevent undesirable actions [36, 18]: in a tower cab position (Blagnac), in case of potential wake turbulence after take-off (or landing), the controller encodes a delay condition for the next take-off by leaving the departing strip (of the aircraft causing the turbulence) *above* the next (ready to depart aircraft), in order to prevent any handwriting on it, and thus any take-off authorization. We also observed numerous bimanual interactions, for example during an approach instruction session where the planner held bundles of strips for each stack in each hand (Figure 7).

*Negative properties*

Tangibility also shows some limits [15] - that can sometimes be addressed by virtual features - such as strip fixed size, static stripboard structure, lack of space or manual operations. Lack of space is a recurrent concern when the traffic becomes dense, despite the fact that the stripboard is well organized, and although the small physical size of the control position enhances mutual awareness [25]. For instance, in a tower cab context, a controller piled strips so as to gain space, and another controller complained about badly designed paper forms that takes too much space: «*Paper, paper, ... a purely electronic form would be nice. In Blagnac, we currently have a paper sheet to fill in, with 60 lines for each minute... half of the sheet is useless. It takes room pointlessly.* ». During design workshops, lack of space was addressed in different ways, for instance through the idea of extensible mini-strips. Controllers also complain about some physical manipulations. In particular, moving groups of strips is tiresome, as may happen when encoding evolving N-S or E-W flight streams on the board, as in the Bordeaux en-route center. In this center, the stripboard has grooves that let the strips be moved together, although in one direction only (to the top). We designed and discussed a bi-directional board with controllers in a workshop (Figure 8a), and their reaction was enthusiastic: « *This thing that goes up and down, yes, I can't wait to get this!* ». Concerning handwriting, one of them said: « *Let's talk about writing the time [on strips]. It takes time, it's heavy, it's a pain!* » This controller thus suggested to replace handwriting with speech recognition. Other problems with physical objects that we noticed and that we could address through augmentation include access to distant strips difficult to reach without disturbing or accidentally moving them, or slippage, that may happen with one-handed writing (Figure 8b) [23].

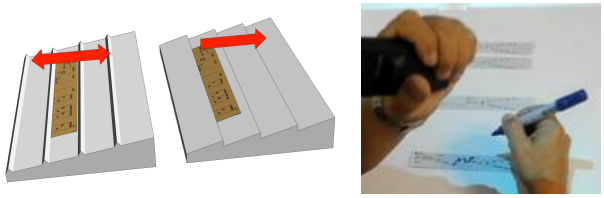


**Figure 8: a) bi-directional grooved stripboard; b) one-handed writing when holding a microphone.**

### Managing mixed interaction complexity

We were concerned that augmentation, mixing physical and digital laws, brings complexity, and thus had to be carefully designed for the users to understand how it works [26, 13]. Regarding this matter, we observed a number of issues, but we also noticed interesting non-issues.

*Issues*

Some controllers were concerned about the "digital consequences" of their once easy to understand physical actions [3]: «*Does the system understand what I'm doing? How do I know? I'm writing something… what can happen in case it's not recognized?* […] *You have to be aware that by mixing electronic and physical systems, you will have a more complicated communication between each… this makes me anxious… it's going to be ultra-complex.* ». Mixing physical interactions and virtual results may lead to discomfort. We walked with controllers through two prototypes (video and PowerPoint) of a tangible computation of an arrival sequence, which can be heavy in some settings having several stacks, such as in Orly (Figure 2): when a strip is laid on a special area of the board

displaying expected final times, the system projects the corresponding stack exit time (Figure 12d). While the controllers found the idea useful and proposed several improvements in a quite participative way, one of them had some difficulties with this simulation area. While thinking aloud about the interface components he was looking at in the video prototype, he said: «*I have to forget this, for us … you need to get this out of my mind… [while hiding his eyes from this part of the board]* …». Unpredictable behavior may also result from some strips manipulations, such as strips askew (Figure 9a) or superposed: in the latter case, as illustrated by Figure 9, the system « works», i.e. virtual strips are projected, but more or less unrelated to the underlying strip. As advocated by [13] such issues have to be explored thoroughly and dealt with by the system, even if the manipulations, as the ones we mentioned, are unusual.

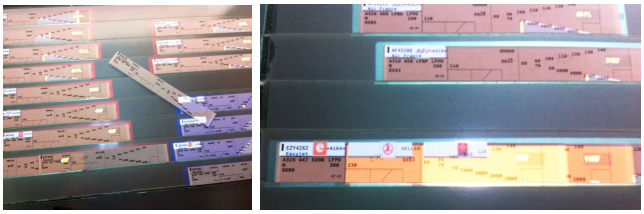


**Figure 9: a) a strip askew; b) superposed strips.**

*Non-issues*

By contrast, several concepts related to mixed interaction were quite easily accepted. Notably, all controllers played with the concept of virtual strips in several ways. They all appreciated the virtual strip as the visible counterpart of the physical strip laid onto the stripboard. To them, it is the main feedback that shows that the system understands what they are doing in physical space: « *It works, and this is the interesting point, that the system knows what we do.* » Feedback is probably one of the most important functions of augmentation (Figure 6, Figure 10). Beyond that, the main outcome is that this « understanding » from the system may bring support for detecting potential problems, such as warning about wrong written clearances or degroupment suggestions according to a growing number of strips detected on the board; warning from the system about possible missing actions, as played by a controller: « *Hey, you keep moving your strips but nothing has been written for a while, what's going on?* » These spontaneously proposed features show the importance of a « mutual understanding »: users need to understand the system, and to know that the system understands their actions.

As for the virtual strips and their physical counterparts, other facts struck us: controllers were quite comfortable with the isolated virtual strips – projected strips not corresponding to any « true » physical strip. We understood that these « informational » strips stand mainly for them as awareness during transient or temporary states (e.g sector degroupment): « *Indeed, having the virtual strip and the data on the real flight… it's just a matter of timing; if he [the controller of the adjacent sector] calls, that will save us some time!* », or for flights that controllers do not have to manage officially, such as flights transiently crossing the sector or very small tourist planes.

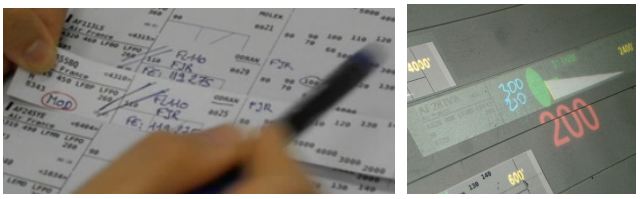


**Figure 10: importance of feedback. a) MOD indicates that this strip is a modified and reprinted one; b) feedback for a handwritten heading.**

Virtual strips were spontaneously proposed for incoming flights, too. To explain this acceptance, we had several explanations. One was that electronic strips are becoming more familiar in ATC culture. Another is that controllers clearly distinguish between « true » official strips that represent their individual responsibility and that belong to the whole ATC system with its flight plans, on one hand, and on the other hand, informational elements that belong to their own view of the traffic and to their own workspace. Another striking fact was that, under certain conditions, controllers did not bother having physical strip duplicates, either as re-printed strips for a given flight, or having a printed counterpart of a virtual strip. What could appear as complex, potentially leading to inconsistencies, in fact did not. This situation seemed in fact acceptable as long as the reprint of the strip is requested by the controller themself, or if the system informs about the status of a reprinted strip (Figure 10).

As for any systems, mixed systems need consistency. Users exploring the prototype during workshops insisted on how confident they are with the homogeneous space that is provided by the system, where all the interacting areas (screen, stripboard, paper) work the same way, with the Anoto pattern and digital pen: « *You have built a unique system for the radar screen and the strips, this goes toward harmonization, this is the way to go.* » Notably, the system enables users to interact with the strip even when it is removed from the board, which is not possible with an interactive surface. This is essential [34]: controllers often take one or more strips in their hands, and point onto the paper with the pen, while either staring at the screen or standing next to the control position, and discuss the current situation (Figure 11a). In this setting, tracking is no longer available, so that projection is understandably disrupted, but pen input and control still works (Figure 11b).

[26] warns about a digital system presenting either too much or too little information to the controllers, arguing also that the physical strips let controllers themselves adjust how much or how little of their mental representation is off-loaded into the strips through annotations and spatial manipulations. In an augmented setting, this physical adjustment is still possible, but we were aware that augmentation should not spoil this positive aspect, and that « just enough » digital information should be added onto the physical objects in order not to increase reading time and interpretation and their potential safety implications. Virtual objects of the prototype

do not have the same status in this regard: indeed, bottom projection can be occluded by paper strips, while top projection cannot. Interestingly, paper cannot occlude top projection either, which may lead to positive effects, when critical information, such as alarms, have to be visible, in as much that top projected objects are not opaque. By contrast, bottom projection is best suited for informative, less critical information that can be displayed in the strip extension.

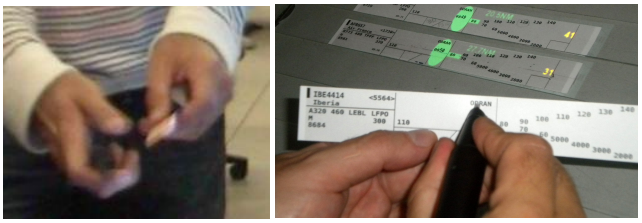


**Figure 11: a) Controllers holding paper strips and pen, and pointing onto it while discussing with other controllers standing in the control room; b) using Strip'TIC: the strip is still digitally interacting with other strips laid on the board.**

### Choosing between virtual or tangible interactions to support ATC temporal processes

This last part reports on more specific design issues related to the support of temporal processes. Current system developments in ATC such as [1] provide tools that use time-based information to manage trajectories. Maestro [2] already provides the controllers in Roissy with a tool to compute their arrival sequence according to explicit time slots. We explored whether augmentation could provide useful support to time-related features.

#### *Structure of temporal processes in ATC*

For air traffic controllers, safety means managing real-time events: planes arrive at their destination or take-off at given times. A critical part of the controller's task is to manage these events in real-time by talking to the pilots to give clearances. Another critical task is full preparation in order to ensure that these real-time actions will unfold properly and effectively. [22] describes ATC activity as a subtle combination of two modes of control: a proactive mode that consists – often for the planner – in building an efficient encoding of the problems so that they can be resolved very quickly and without errors, and a reactive mode which is often performed by the tactical through reactions to events (pilot calls, clearances, potential conflicts).

The two parallel modes occur at different timescales, as explained by a controller: «*the tactical […] is dealing with a problem at 15 nautical miles and we speak here about a conflict that will happen in 15 minutes*». Proactive mode is related to data encoding (flight integration, arrival sequence preparation), transition management tasks (sector degroupment, team replacement), and also problem encoding (filtering, searches, annotations, strip specific layouts). Reactive mode, characterized by fast context switches where data for problem solving must be at hand, is related to actions and decisions through physical gestures and tangible artifacts. ATC activity thus involves constant phasing between two timescales: that of the controllers, during which they organize their work, and real time, where real traffic occurs.

#### *Temporal processes physical encoding*

In [18], Kirsh describes how spatial arrangements support expert activities involving a preparation phase, and a high tempo execution phase. For instance, experts ensure that information needed to act quickly is available locally, and that actions can be performed almost automatically. To achieve this, they pre-structure their workspace physically to simplify choices, discard unrelated decisions, encode pre-conditions, and highlight the next action to take. For Kirsh, space also naturally encodes linear orders: items arranged in a sequence can be read off as the order they are to be used in. We observed similar orders in approach control with stacks and arrival sequences. In [10], Harper et al highlight how these arrangements encode an ordered set of tasks to perform: "ATCO work is not like an assembly line in which a recurrent sequence of steps has to be followed through, but one in which the work consists of putting the tasks to be done into a sequence of steps that can be followed through. […] This is how the ATCO is looking at the information presented in the strips, the radar and the R/T; to see what needs doing 'now', 'in a moment', 'sometime later on', and so on". These spatial orderings implicitly connect the two timescales we mentioned above: taken as traffic sequences, they correspond to the planes flying in real-time, but as tasks to perform, they also correspond to the control timescale. To be as precisely on time as possible, controllers also rely on their knowledge of action duration according to various contexts (including their own cognitive load): «*It's your internal clock, you know how long it takes you to perform standard actions.*»

#### *Experimentation of time-based mixed tools*

Based on this analysis, we explored how to turn these implicit relationships into a more explicit design of virtual temporal objects. We designed and implemented several prototypes, where time-related information was provided for various purposes, for instance to help calculate a stack exit time (Figure 12d). This was inspired by a kind of paper ruler that is used by Orly approach controllers as rough paper that helps visualize free time slots and calculate mentally (Figure 12c). We also implemented a tool to compute the time to reach a beacon (Figure 5) or to fly a given trajectory drawn as a polyline on the radar. In addition, we designed a timeline representing several flights heading to a common beacon [9] to analyze potential conflicts. Such tools are meant to add explicit time-related information to the already spatially structured linear orders. Time can also be visualized as dynamic, providing a sense of passing time through information that evolves visually, such as a timer to manage wake turbulence (Figure 12b), or progress bars (Figure 12a). We see these tools as complementary instruments to support phasing between the two timescales that we described above. What we observed however is that controllers quite efficiently rely on their own skills using physical and spatial tools both to adapt to real time and to schedule their actions.

In anticipation mode, approach controllers are in fact not so much interested in explicit time, but rather on ordering: «*You don't care about arrival time, what matters is that they [the planes]… are in a sufficiently spaced out and coherent order… not too close, not too far…* ».

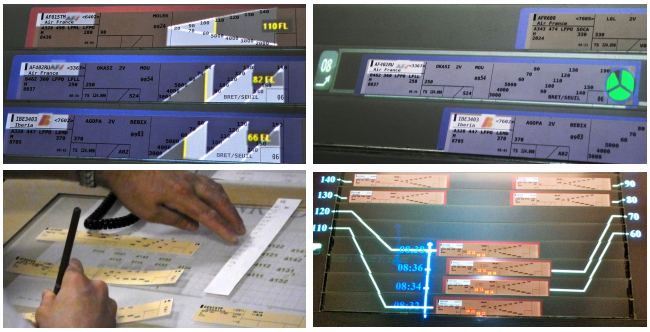


**Figure 12: a) colors (light grey, yellow, dark grey) indicate past, present and future flight levels; b) wake turbulence take-off timer (green circle): the next departing plane has to wait for the AFR608 (indicated as heavy (H)) to move away; c) a rough paper ruler on the back of a strip to allocate an arrival sequence; d) tangible computation of an arrival sequence: allocation within projected time slots (blue), computed stack exit times (red), strips linked to their projected stack level (white - crossing links show a misordering).**

One controller was in fact more interested in dynamic augmented features, as long as they are real-time, tactical control oriented, and help program timing or actions. While we were discussing an arrival timeline, he spontaneously proposed the idea of a countdown timer [4] to trigger action reminders: « *…10 ...9 …8 …0 …-1 …-2 … something to remind the tactical controller that it's time to act, to give an order to the pilot, and then even how much he is behind.* » This type of timer links control time and real-time by supporting the controllers in scheduling their actions.

## DISCUSSION

In this study, we have chosen a tangible interaction perspective to analyze our observations, rather than an augmented paper one. As argued by [37], paper-based interfaces can be considered as TUIs, since they provide users with a physical modality of interaction and propose a close mapping between input and output. In addition, paper strip "thingification" makes them more relevant as physical cardboard handles, than as paper documents. Finally, a reason for adopting our perspective is that tangible interaction provides design models for coherence, that we wanted to investigate to address mixed interaction complexity. In this section, we first reflect upon our observations in terms of coherence, and more specifically in terms of representation. Then we describe how Strip'TIC addresses complexity through a seamless interactive space.

### Challenging mapping and metaphorical coherence in TUIs

#### *What do physical objects actually represent?*

To date, as noticed by [20], one of the main stated concerns of tangible interfaces is how representative a physical object is of a virtual one. This explains why a mouse cannot be considered as a tangible interface: it does not represent any object of interest [35]. In [7], Fishkin describes tangible interfaces according to how closely physical input and virtual output are tied together (embodiment dimension) and how similar they are (metaphoric dimension). Unless used as tools, Holmquist [11] also describe physical objects as representing digital objects: a container potentially represents any virtual object, while a token stands for it. In these approaches, it seems that representation must be understood as both a statement of likeness and one of semiotics, where the physical object behaves as a *sign*, i.e something that stands for something else [35].

During the design of Strip'TIC, we were faced with this representation issue in a slightly different manner. What do physical strips actually *stand for* in this environment? For the controller, physical strips stand for flights crossing their sector and for their associated responsibility. They do not stand for virtual strips: the bottom projected strips mostly act as feedback, not as objects to manage. They do not stand for the flights displayed on the radar screen either: tracks on the screen and strips are different objects serving different purposes. The radar screen provides a view of real-time traffic, whereas the stripboard represents traffic and task achievement. Pointing onto a strip does in fact select the corresponding aircraft, but this just provides a visual transition between complementary views.

In the previous section, we have described spatial arrangements as an encoding for a set of control actions to perform, or for problems such as conflict detection. The physical layout and associated handwritten annotations of strips provide a structure that helps coordinate thoughts and build an image of the state of the system. What Kirsh in [18] describes as external representations enable memory to be offloaded (as stated for strips in [25]). In addition, they help to build persistent referents to internal information, that can be rearranged to perceive aspects that were hard to detect and to improve perception of semantically relevant relations [18].

#### *Virtual objects representing physical objects*

Virtual strips deserve a separate comment regarding their representational status. During phases where physical strips are missing on the control position, such as degroupment, it is the *virtual* strip that stands for the physical one, and the controller can interact with it as if it were the physical one. This status is important because it shows that virtual strips provide redundancy and thus robustness in cases of absent strips. The metaphorical expressivity of virtual strips also builds on prior cultural knowledge [15] that controllers have gained on electronic strips, as described in the previous section.

### Toward a convergent design to support externalization through tangible interaction

#### *Physical interactions supporting externalization and control*

So, while physical strips do not seem to stand for any virtual object, they do stand for a responsibility for an aircraft and act as external representations of this

responsibility. At the same time, Strip'TIC provides a true tangible space, i.e. a space where manipulations in physical space trigger events in the virtual space: moving strips on the board moves the associated projected data. Therefore, we can identify the two relationships described by the MCPrd model [35]: the physical strip *controls* digital data projection, but *stands for* an internal representation. On one hand, the physical strip acts as a tangible "window" to control the output projected onto the paper (Figure 13b). On the other hand, the physical strip acts as a cognitive handle to project and build [19] a mental picture (Figure 13c). Through physical manipulations, each of the two dimensions builds an image: a projected image, and an internal image of the current state of the situation. It should be noted that the projected image comes at no cost: controllers are not aware of this "window management" activity.

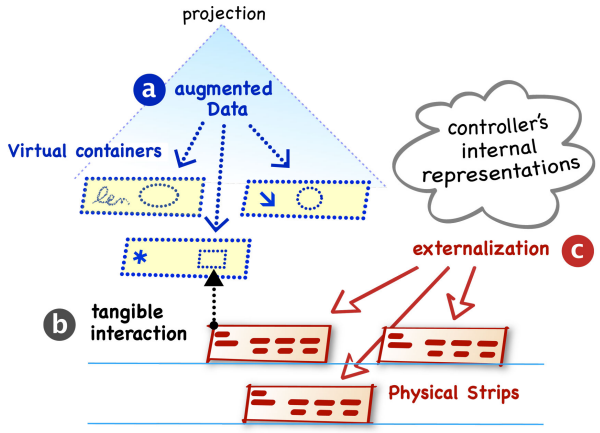



**Figure 13: tangible strips (b) acting as *controls* and as *containers* for augmented data (a) and as *representations* and *tokens* for cognitive elements (c).**

*Implications for allocation of physical and virtual components*

This analysis sheds light on our choices of allocation. As we described in the previous section, physical objects and associated manipulations have their inner coherence. So, as long as physical objects are able to provide the controller with external representations of their concerns, there is no need to overload them with additional explicit information. As described in the previous section, physical and spatial tools provide a sufficient encoding of objective time, orders and internal clock. By contrast, augmented data (Figure 13a) are needed to provide real-time perception and dynamic information on the current state of flights. Our analysis also helps to understand potential complexity issues, where the physical manipulations, such as tangible computation of stack exit times, did not exactly correspond to current practices, i.e., at least for some controllers, to externalizations on which they rely today.

### A seamless and understandable interactive space

Complexity is also dealt with through the properties and the behavior of the components. Several of our observations highlight this aspect, both in terms of interaction devices and of human-system communication [3]. Controllers often commented on the homogeneous Anoto patterned environment, providing a uniquely addressable system. They also reacted particularly well to the system showing a « mutual understanding » through constant attention to user input (strip moves, handwriting recognition) and continuous feedback. Continuous feedback notably addresses issues discussed by [26] and [13], such as user understanding and expectations about the system behavior. The system shows additional kinds of continuity: 1) in the strip itself, which can be described as an *inherent mixed object*, combining a physical nature (paper) with a virtual content (printed information which can be re-printed on demand). This provides a mixity that might blur the frontiers between physical and virtual, as advocated in [31]. When data are projected onto the strip surface, a fine-tuning of luminosity may produce the effect of a composite rendering of printed and projected data. 2) merged displays: projected data cannot occlude printed information and vice versa, which adds to this seamless combination of tangible and virtual dimensions. Finally, coherence in Strip'TIC does not assume a constant coupling between physical and virtual components: disrupting strip tracking by removing a strip from the board does not prevent continuous use of the system, and most importantly, disrupting Anoto does not break the system either, since handwriting still works.

## CONCLUSION

The Strip'TIC system provides us with the ability to explore mixed interaction in a context where physical interactions are effective and secure. Throughout our design reflections, we have seen that unlike usual TUI approaches, which rely on mapping and coupling of physical objects to virtual objects, coherence in Strip'TIC is based on other aspects. First, it relies on the mapping of *virtual to physical* objects that play a primary role as externalization of controllers internal concerns. Second, coherence consequently relies on a seamless connection of cognitive and tangible spaces that help the users to build a physical *and* virtual image of the situation. Third, the properties of the interactive space components and continuous feedback help users understand the mixed behavior of the system. Thus, compared to existing TUI models, our approach is complementary: we include the understanding and integration of artefact cognitive mechanisms as part of our design choices. Future work involves exploring issues about how augmented data may support externalization too, since this matter seems to be overlooked in current research. We also plan to investigate further about multitouch gestures combined with handwriting and pen-based interaction.

## ACKNOWLEDGMENTS

We thank the air traffic controllers for their participation to the workshops, M. Cordeil for his help on coding and C. Savery and R. Moreau for the work on the video.

## REFERENCES

1. Aviation system block upgrade modules relating to trajectory-based operations, Working paper, in 12th Air Navigation Conference, ICAO, Montreal, may 2012.

2. Benhacene, R., Hurter, C., Marion, A., Merlin,B., Rousselle, M.P., Ribet, P., ASTER, tackling the problem of flight integration. 7th ATM seminar Barcelone 2007.
3. Bellotti, V. Back, Edwards, M. W. K., Grinter, R. E., Henderson, A. and Lopes, C. Making sense of sensing systems: five questions for designers and researchers. In *Proc. of* CHI '02. ACM.
4. Conversy, S., Gaspard-Boulinc, H., Chatty, S., Valès, S., Dupré, C. and Ollagnon, C. 2011. Supporting air traffic control collaboration with a TableTop system. In *Proc. CSCW '11*. ACM.
5. Fallman, D., Design-oriented human-computer interaction. In *Proc. of* CHI '03. ACM.
6. Fernaeus, Y. and Tholander, J. Finding design qualities in a tangible programming space. In *Proc. of* CHI '06, ACM.
7. Fishkin, K.P. (2004). A Taxonomy for and Analysis of Tangible Interfaces. *Personal and Ubiquitous Computing*.
8. Fitzmaurice, G., Ishii, H., Buxton, W. Bricks: laying the foundations for graspable user interfaces. In Proc. of CHI'95. ACM
9. Grau, J.Y., Nobel, J., Guichard, L. and Gawinoski, G. Dynastrip: a time-line approach for improving the air traffic picture of Controllers,". In *Proc. of DASC '03*.
10. Harper, R.H.R. and Hughes, J. A. (1991) What a f-ing system! Send 'em all to the same place and then expect us to stop 'em hitting": Making Technology Work in Air Traffic Control, Technical Report EPC-1991-125.
11. Holmquist, L., Redstrom, J., Ljungstrand, P. Token-based access to digital information. In *Proc. of* HUC'99.
12. Hornecker, E., Buur, J. Getting a grip on tangible interaction: a framework on physical space and social interaction. In *Proc of* CHI '06. ACM.
13. Hornecker, E. 2012. Beyond affordance: tangibles' hybrid nature. In *Proc. of* TEI '12.
14. Hurter, C., Lesbordes, R., Letondal, C., Vinot, J.L. and Conversy, S. Strip'TIC: exploring augmented paper strips for air traffic controllers. In *Proc. of* AVI '12, ACM.
15. Jacob, R.J.K, Girouard, A., Hirshfield, L.M, Horn, M.S, Shaer, O., Solovey, E.T. and Zigelbaum, J. Reality-based interaction: a framework for post-WIMP interfaces. In *Proc. of* CHI '08. ACM.
16. Jorda, S., Kaltenbrunner, M., Geiger, G. and Bencina, R. The Reactable. *In Proc. of* ICM 2005.
17. Kirk, D. Sellen, A., Taylor, S., Villar, N. and Izadi, S. Putting the physical into the digital: issues in designing hybrid interactive surfaces. In *Proc. of BCS-HCI '09*.
18. Kirsh, D. (1995) The intelligent use of space. Artificial Intelligence. *Artif. Intell.* 73, 1-2 (Feb. 1995), 31-68
19. Kirsh, D. Interaction, External Representation and Sense Making. In *Proc. of the Cognitive Science Society*. Mahwah, NJ: Lawrence Erlbaum. 2009.
20. Klemmer, S.R., Hartmann, B., Takayama, L. How bodies matter: five themes for interaction design. In *Proc of* DIS '06.
21. Koleva, B, Benford, S, Kher Hui Ng, Rodden, T. A Framework for Tangible User Interfaces, Physical Interaction (PI03) - Workshop on Real World User Interfaces, Mobile HCI Conference 2003.
22. Loft S, Sanderson P, Neal A, Mooij M., Modeling and predicting mental workload in en route air traffic control: critical review and broader implications. Human Factors. 2007 Jun; 49(3).
23. Luff, P., Pitsch, K., Heath, C., Herdman, P., Wood, J. (2010) 'Swiping paper: the second hand, mundane artifacts, gesture and collaboration' *Personal And Ubiquitous Computing*, 14 (3).
24. Mackay, W.E., Fayard, A.L., Frobert, L., Médini, L., Reinventing the familiar: exploring an augmented reality design space for air traffic control. In *Proc. of* CHI '98.
25. Mackay, W.E., Is paper safer? The role of paper flight strips in air traffic control. *ACM Trans. Comput.-Hum. Interact.* 6, 4 (December 1999).
26. Mackay, W. E. Augmented reality: dangerous liaisons or the best of both worlds? In *Proc of* DARE '2000.
27. Manches, A., O'Malley, C. and Benford, S. Physical manipulation: evaluating the potential for tangible designs. In *Proc. of* TEI '09.
28. McGee, D.R, Cohen, P.R., R. Wesson, M. and Horman, S. 2002. Comparing paper and tangible, multimodal tools. In *Proc. of CHI '02*.
29. Mertz, C., Chatty, S. and Vinot, JL. The influence of design techniques on user interfaces: the DigiStrips experiment for air traffic control. HCI-Aero 2000.
30. Rekimoto, J., Ullmer, B. and Oba, H. 2001. DataTiles: a modular platform for mixed physical and graphical interactions. In *Proc.* CHI '01. ACM.
31. Robles E. and Wiberg M. 2010. Texturing the "material turn" in interaction design. In *Proc.s of* TEI '10. ACM.
32. Sellen A.J., Harper R.H.R. The Myth of the Paperless Office. MIT press, Cambridge, MA. 2001.
33. Signer B., Norrie M., Interactive Paper: Past, Present and Future. In *proc of* UbiComp'10.
34. Terrenghi, L., Kirk, D., Sellen, A., and Izadi, S. 2007. Affordances for manipulation of physical versus digital media on interactive surfaces. In *Proc. of CHI '07*.
35. Ullmer, B. and Ishii, H. 2000. Emerging frameworks for tangible user interfaces. *IBM Syst. J.* 39, 3-4 July 2000.
36. Zhang, J., Norman, D. A. (1994) Representations in Distributed Cognitive Tasks, Cognitive Science 18(1).
37. Zufferey, G., Jermann, P., Do-Lenh, D. and Dillenbourg, P. Using augmentations as bridges from concrete to abstract representations. In *Proc. of* BCS-HCI '09, UK.

# Designing Graphical Elements for Cognitively Demanding Activities: An Account on Fine-Tuning for Colors

Gilles Tabart[1,3], Stéphane Conversy[1,3], Jean-Luc Vinot[2], and Sylvie Athènes[4]

[1] LII ENAC, 7, avenue Edouard Belin, BP 54005, 31055 Toulouse, Cedex 4, France
[2] DSNA, DTI R&D 7, avenue Edouard Belin, BP 54005, 31055 Toulouse, Cedex 4, France
[3] IHCS IRIT, 118, route de Narbonne, 31062 Toulouse Cedex 9, France
[4]EURISCO International, 23, avenue E. Belin, BP 44013 31028 Toulouse Cedex, France
tabart@cena.fr, stephane.conversy@enac.fr, vinot@cena.fr, sylvie.athenes@eurisco.org

**Abstract.** Interactive systems evolve: during their lifetime, new functions are added, and hardware or software parts are changed, which can impact graphical rendering. Tools and methods to design, justify, and validate user interfaces at the level of graphical rendering are still lacking. This not only hinders the design process, but can also lead to misinterpretation from users. This article is an account of our work as designers of colors for graphical elements. Though a number of tools support such design activities, we found that they were not suited for designing the subtle but important details of an interface used in cognitively demanding activities. We report the problems we encountered and solved during three design tasks. We then infer implications for designing tools and methods suitable to such graphical design activities.

**Keywords:** User interface, graphical rendering, graphical design, color design, design study, critical systems.

## 1 Introduction

Visualizations of rich graphical interactive systems are composed of a great amount of graphical elements. Perception of graphical elements is highly dependent on multiple interactions between visual dimensions such as color, area, shape etc. and display context such as type of screens and surrounding luminosity. Understanding these interactions involves multidisciplinary knowledge: psychophysics, human computer interaction, and graphical design. How can visualization designers make sure that they minimize the risk of confusion? How can they be sure that any modification done on a 20 years old system will not hinder the perception, and hence the activity, of the users? How to convince users and stakeholders? In general, how can they design, validate, check, assess, and justify their design?

This kind of questions has been addressed at the level of the design process for the functional core, with methods such as Rational Unified Process or with Design Rationale

T.C.N. Graham and P. Palanque (Eds.): DSVIS 2008, LNCS 5136, pp. 136–148, 2008.

tools [9], or at the level of code, using tools based on formal description of interaction, such as Petri Nets [1]. However, tools and methods to design, justify, and validate user interfaces at the level of graphical rendering are still lacking. A number of past studies addressed this problem, but their results did not quite apply to the specific kind of user interfaces we design: those that contain multiple, overlapping elements, the perception of which are very dependent on subtle details, and that users scrutinize during long periods of time in a demanding cognitive context. Good examples are the latest generation of jetliners, in which pilots interact with graphical elements on liquid crystal displays (LCD) to manage the flight, or air traffic controllers who rely mostly on radar views with multiple graphical elements, to space aircraft within safety distance. As these interactive systems are used in critical situation, the need for sensible, justified, and verified design is even more important.

In order to design such tools and methods, one must identify the relevant dimensions of the activity that they are supposed to support. This paper is a report of graphical design activities for interactive systems. We present our experience as designers during various design activities we conducted. We then discuss important considerations one has to take into account during such activities, or if one wants to design tools and methods to support it.

## 2 Related Work

Graphical design issues have been studied by organizations like W3C [16], FAA [7] and NASA [13]. They have established a batch of guidelines about UI graphical design and recommendations about common visual perception issues. Researchers in information visualization worked on efficient representations [5,17]. Graphical semiology introduced visual variables (size, value, color, granularity, orientation and shape) together with their ability to present nominative, ordered or quantitative data [2]. Brewer [3] proposed tools to help design harmonized color palettes for cartography visualizations. Lyons and Moretti analyzed current color design tools [11], and designed a tool for creating structured, harmonious color palettes [10]. We extensively used guidelines from NASA and Lyons & Moretti molecules approach. They help guide the design process, and help structure the colors used. However, NASA guidelines are short on precise guidelines with subtle but important rendering problems. In addition, the molecules tool does not provide much help for the kind of constraints and needs we had during the process.

## 3 Studies and Experience Feedback

In this section, we present three design tasks that we conducted. We redesigned interactive systems that support air traffic controllers. In order to understand the design process, we first set the context by briefly presenting air traffic control (ATC) and the three tasks we had to accomplish as designers. We then report on our experience.

### 3.1 Air Traffic Control Activity

All our tasks dealt with graphical design issues pertaining to the main French radar screen software used by the air traffic controllers. The software main goal is to display

three-dimensional aircraft positions as if seen from above. The air space is divided into "sectors": complex three-dimensional airspaces criss-crossed with various routes. Each sector is managed by a team of 2 controllers: the tactical controller, who monitors aircraft through the radar screen and give vocal orders to pilots through a radio link, and the planning controller who organizes flights arriving from neighboring sectors. Controllers rely on flight plans, requests by pilots, requests from other sectors, current weather and traffic conditions to manage the air traffic, making judgments about the most efficient and safe way for aircraft to proceed through the air space while keeping within safe distance from each other. Each controller faces a radar screen displaying the sector under his/her responsibility. Each aircraft is represented as an icon showing its current position and smaller icons showing a few of its past positions. The current position is linked to a label with the flight identifier, current speed and flight level. In accordance with the controller's preferred settings, each screen might have a different configuration (zoom level, pan, visible layers, etc).

In ATC, the graphical information displayed has a high level of criticality. A controller may hold the fates of several thousand people during his work shift and his judgment is based on well-established work practice, his experience, and his perception of the displayed information. Therefore, all information has to be coherently displayed, in a very accessible but not intrusive fashion in order to spare the cognitive resources.

### 3.2 Design Process

As the tasks are mostly concerned with designing colors, we present the approach we used in terms of color model, tools, and methods.

*Color models, calibration, and tool*
RGB is the color model used in graphic computer-cards for encoding color. RGB is based on additive syntheses of colors using 3 primaries: red, green and blue. Software developers often use this model to specify color. RGB is a "machine-based" model: it is difficult to manipulate, and hinders the structuring of color choices. Other color spaces, such as models proposed by the Commission internationale de l'éclairage (CIE, International Commission on Illumination) and specially CIE LCH(ab) are "human perception-based" model. We used the LCH color space for two reasons. As LCH is a mostly linear perceptual model, it allows predictable manipulations. Furthermore, the L (luminosity), C (saturation), and H (hue) dimensions are semantically known color dimensions which further structured design: it helps organize colors (and hence conceptual entities) with three mostly orthogonal dimensions. In the remaining of the paper, we call "color" the perceptual phenomena referring to a particular LCH or RGB combination (and not simply the hue). We applied a calibration process on each monitor we used, so as to minimize the effects of bad rendering chain settings. Furthermore, we defined a reference ICC profile [8], and used it while designing colors, so as to maximize consistency between design sessions.

During our tasks, we designed and used our own tool to choose and modify colors. The tool can import a set of colors, sort colors into group, and display them around a hue circle using the LCH model and the reference ICC profile. It also allows to express constraints with "molecules" of color [10], or to modify directly their hue, their level of saturation, or their level of luminosity (Fig. 1). We will not describe further this tool, as it is not the purpose of this paper and is only a draft of what should

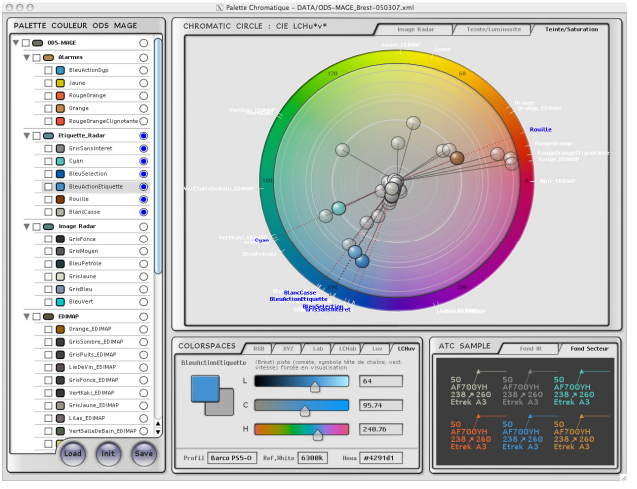



**Fig. 1.** The ad hoc tool with the color palette, color wheel, color spaces and color samples

become a genuine instrument. However, it helped us identify relevant aspects about the design activity and about desirable features of an efficient tool.

*Context of the design*

During the first task, we designed colors directly in the control room. We had to work on specific displays that were installed in control centers in order to design with real activity conditions in mind. In addition, we had to take the controllers' opinions into account and iterate with them to reach an agreement and validate our work. As previously said, a control position comprises two screens. We kept an image of the old configuration on one display and applied modifications to the other so that we could compare the results of the transformation and discuss them with the controllers. We also displayed the old configuration on the old CRT monitors to compare between color renderings. Using an actual configuration also allowed us to check if looking at the screen from different visual angles did not influence too much color perception.

The colors were then translated to RGB and inserted in the radar view configuration file, in which color names are matched with their RGB hexadecimal code, e.g. `(name "Orange") (value 0xd08c00)`. When drawing a graphical element, the software refers to colors by their name e.g. `ConflitEtiquette#N_Foreground: MC#Orange#NColorModel`. Using this indirection, designers can share the same color between different elements. For example, when an alarm has to be applied both to a radar track and to an information panel (Fig. 2}, a designer can tag these two elements with the named color *orange*. Thus, if the hexadecimal value of orange color is modified, all orange elements will be changed. The configuration scheme is a way for the designers to structure color-coding. As such, it makes the task of configuring the radar view easy, and enables the system to accommodate unexpected changes or important security fixes. For the two other tasks, we worked on our computer on which we imported the palette to be changed.


**Fig. 2.** Two elements: same color code but not identically perceived

### 3.3 Design Activities Study

Our team includes a graphic designer, an experimental psychologist, and two HCI specialists. The tasks we present are real-world tasks: they are part of an industrial process, as changing such systems must follow precise steps. We were then constrained in the amount of modifications we were able to recommend.

*First task: updating a global color design*
Our task was to adapt the color settings of the main radar view software. presents the interface: the control panel on the left side, the main radar view in the middle, and the flight lists on the right side. The left panel present manifold options for choosing pan and zoom level or slices/layers of the sector to displayed, for example. On the main view, different areas are represented in the background with different colors, while 1 pixel wide lines represent flight routes. Flights current and past positions are represented by 3 to 5 pixel wide squares. A tag with textual information about the flight (callsign, level, speed etc.) is linked to the shape with a 1 pixel wide line. The right panel is reserved for alarms and a list of flights. Selected flight information is displayed at the bottom of this panel.

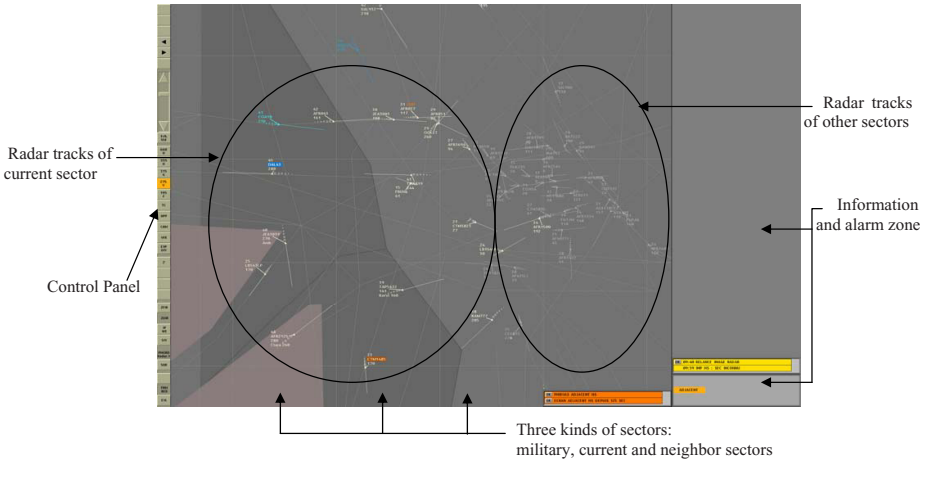



**Fig. 3.** The main visualization for air traffic control

We had to adapt the color settings because the system evolved both technologically and functionally. The CRT displays on which the application runs have been replaced with LCD displays. Color rendering on LCD differs from CRT: they are more saturated, while the beam is narrower. The difference in rendering completely changes the overall appearance of the visualization. Furthermore, the CRT displays are square while LCD displays have a 16:10 ratio, which changes the proportional amount of the different graphical elements. Beside hardware evolution, the activity regularly evolves, with the addition of new functionalities, new control procedures or new sector arrangements. This results in stacked modification, with no real global design.

On the one hand, we had to hold the perceived color constant while moving from CRT and LCD display. On the other hand, we had to harmonize color palettes between configurations from five air traffic control centers. Each one has its own color palette, due among others to traffic particularities. This specific task may seem trivial (changing colors); but to achieve it, we had to modify almost all colors of the application, and a lot of questions and problems were raised.

*Second task: organizing flights into categories*

The second task was to add new colors to an existing color palette. This requirement came from a new need in approach control activity. Controllers doing "approach control" regulate air traffic around airport areas. They needed to distinguish three categories of flights around Paris, those concerning Orly airport, those concerning Roissy airport, and in transit flights. They also needed to separate flights into two flows, e.g. Orly or "Orly-associated" airports. Together with users, a team of engineers had previously designed and installed a palette with three named colors ("green", "pink", "blue"). We had to harmonize the palette, while keeping identifiable colors.

*Third task: redesign of an interface*

The third task consisted in the entire redesign of the prototype of a future radar view. We were less constraint by historical constraints, and freer to test original configurations. Even though this task is still in progress, it has brought some valuable information.

### 3.4 Design Accomplishment and Teaching

This section presents a description of our work as designers. The description is organized around the similar issues we encountered during the three tasks. For each issue, we describe our goals, the constraints driving our choices and the solutions we eventually chose.

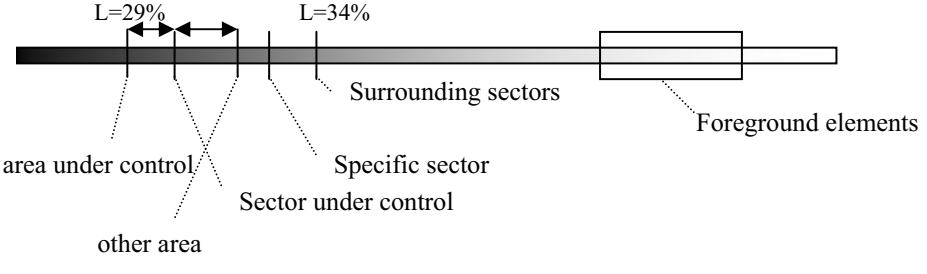



**Fig. 4.** Representation of the graphical elements' luminosity

*Information visibility, luminosity and background*

The first issue concerns information visibility (am I able to see an entity?), discriminability (am I able to differentiate between two entities?), identifiability (am I able to identify an entity among a known set of entities?) and legibility (how easily read is the text?). We first worked on luminosity. All the colors we designed are achromatic: there is no perceptible hue information. In the LHC model, it is implemented by setting color saturation to null.

Luminosity difference enables separation of juxtaposed and layered objects. Sectors are large, uniform surface juxtaposed on the background. Controllers must discriminate and identify them so as to see if a flight is about to enter or leave their controlled sector. Layered objects include sectors (background), routes, beacons, and flights (foreground). Routes and beacons must be visible, while flights must pop out and be legible.

We first designed sectors luminosities, since they end up acting as the background for most objects, and foreground colors can only be set according to the background. Fig. 4 shows the resulting distribution of the gray luminosities of the different sectors: a gray for the controlled sector, another gray for the surrounding sectors, and a last gray for a special area. The sector under control is the darkest: this sector is the most important for the controllers, and flights should be maximally visible here. The surrounding sectors are thus lighter. In bi-layered sector, controllers have to distinguish between two areas in the sector under control. We spread apart two grays around the gray of the sector under control. The second area luminosity is farther from the controlled area luminosity than the surrounding sector luminosity because it is more important to identify the controlled area than the others. However, the second layer gray must also be different enough from the surrounding sector gray. The four grays are very close in luminosity (range L=5%).

This example highlighted a problem with the possibilities of choosing a color in a relatively small range. This issue comes from imposed constraints about gray and from the fact that the 8bit-per-channel RGB color space used by the system is poor; it does not contain enough values to express all the shades of a color range. On a machine color model, grays are made by mixing equally R, G and B. Thus, between white and black there are only 254 possible grays. Furthermore, we precisely tuned the set of grays by incrementing or decrementing RGB values one by one, as the conversion between LCH and RGB was not precise enough. We had to work with the system color space instead of the perceptual color space.

Some graphical objects must be more than simply visible. For example, alarms must grab attention when they are displayed. Even though other visual dimensions such as animation help grab the user's attention, we chose to separate them from background or others grays elements with one additional color dimension, the saturation. Indeed, alarms have specific hues that reflect the emergency level. We gave alarms object a high level of saturation to accentuate the discrimination from background objects.

Some areas, such as military zone, can be considered as an "alarming" area. To differentiate them from civil area and give them an alarming appearance, we decided to slightly color them with a reddish gray.

*Confidence and comfort*

The global image must be harmonious: even if it is difficult to formally quantify it, the satisfaction resulting from using a good-looking image nevertheless matters. Moreover, it improves the controllers' confidence into the system. For example, the planning controller typically configures the zoom level to have a global view of future flights arriving in his sector. However, for narrower sectors, a lot of gray flights not under this controller's responsibility become visible on both sides of the current sector, because the new screen has a 16/10 ratio. These flights tend to raise the object density of the image too much. The global scene perception is spoiled and controllers are less confident in their ability to analyze the image. This resulted in uncomfortable situations, where controllers were afraid to miss an important event, and felt obliged to constantly check the image. This issue never arose with square screens.

Global comfort of the scene is also an issue when designing alarms. On the one hand, alarms must interrupt the user and be remembered, so they are intrinsically not comfortable. On the other hand, if an alarm comes to persist on the screen (e.g. the controllers have seen the alarm but they have to finish some other actions first, or because no action allows the controllers to get rid of them), it should not hinder the controllers'activity. In order to increase comfort with such persistent alarms, we had to decrease their saturation level, and make them less "flashy".

*Categorizing and ordering graphical objects*

One important point in the design process is categorizing and ordering objects. In the second task, we had to group flights in categories and flows. The three main categories had color hues that had been decided in a past design: green, pink and blue. The controllers proposed a separation into two sub-flows. They designed a solution by using various color dimensions, which resulted in heterogeneous colors. We worked with the LCH color space in order to homogenize the design choices. We set apart the three hue angles by 120 degrees and we distributed the sub-flows around each main hue. In order to see the results and finely tune the design, we built an image containing 6 examples of the exact shapes to be colored. We embedded this image in the tools we used, as can be seen at the bottom left of Fig. 1.

We tried to match the conceptual hierarchy with the perceptual hierarchy. For example, the two kinds of flights displayed match their relative importance for the controller. Flights that the controller has currently in his charge are represented in a bright color and others, controlled by neighboring sectors, are in a darker gray.

Alarms are also graduated: according to their importance, they have a certain hue and saturation level. We had to conform to alarm hierarchy and cultural color habits (such as red for danger).

*Surface does matter: perception and software design limitation*

We observed that surface influences perception: according to the surface on which a color is applied, the perception of this color is different. For example, we designed a color for small/medium size military sectors. The color is a gray with a hint of red (which name is "lie de vin"). We later used the color palette in another control center, embedding a larger military zone. When this same color was applied to this surface, the reddish gray seemed too saturated (i.e. too red). We had to decrease the saturation in order to make sectors look grayer when they are big, but still keep a distinctly reddish nuance when they are smaller. Fig. 2 shows a second example with two elements

displayed with the same alarm color code. The first element is a 1 pixel wide text; the second one is the background of an information panel. Due to surface and/or pixel arrangement, the same orange color applied to both these graphical elements does not appear to be the same when a text or a background.

We have been able to accommodate the problem in the first example with a single color. But it proved to be impossible in the second example: we had to design two colors. It follows that the configuration file is not as structured any longer: if one decides to change the orange in the future, one has to change two colors instead of one. This matter is linked to the use of the indirect method for coding colors that we presented above. With a simple indirect color-coding scheme, there is no means to accommodate for differences in color perception due to the amount of surface. This example shows that the coding method can hinder the controller's activity: there is a risk that a color is not identified as corresponding to a particular state, or that two elements cannot be associated through their color.

Another issue concerns very small elements like one pixel-wide lines or glyphs. When we applied low saturated colors to such elements, their hues did not come out very well. These observations can be explained by the fact that, with this kind of small elements, some pixels may end up being isolated on a background color. They are thus "eaten" by background colors and lose some of their properties [14].

*Human subjectivity: naming color, acceptability opinions.*
The next issue is about color perception properties. In the LCH color space we used to organize colors. L, C and H dimensions are supposed to be orthogonal, i.e. if a designer changes a color along a single dimension, the perception of the other dimensions should not change. However, if some colors can be modified in saturation or luminosity without losing their essence (think of light or dark blue), some colors cannot be easily modified without impacting perception of hue. Red for example is identified as such only for a medium luminosity level, otherwise it is identified as ochre/brown with low luminosity, and pink with high luminosity. We experienced this problem when we tried to lower the saturation of alarms, because they were too sharp: when we applied the modification, the element was not perceived as red any longer, but as ochre, which completely disabled its identification as an alarm. We had to change both saturation and hue to keep a color identifiable as red. This phenomenon shows that colors cannot be modified automatically, or at least without precaution.

A related issue concerns the naming of colors. In their activity, controllers use color name so as to identify graphical elements. For example, they use the name of the color to refer to a particular flight status instead of referring to the status itself, as in "can you check the bathroom green airway?". In such circumstances, if a color has to be modified, it must be kept recognizable and identified as the same named color to accommodate historical use.

Human subjectivity is also an issue. For example, there is a large diversity of opinions about the saturation thresholds between a comfortable color and an uncomfortable one. This depends on human perception and sensation but also on the hue value. Furthermore, opinions vary in time, because of habituation or fatigue: the same person can disagree with a design choice at some time, and then agree with it later.

*Display context*

The perception of colors is dependent on the type of monitor. Nowadays, controllers use multiple screens: a radar view, but also a list of flights view, displayed on an almost horizontal screen under the radar view. The colors used on this screen must match the colors used on the radar view, as some of them allow elements to be grouped. However, even after calibration, it proved to be difficult to get exactly the same colors on both screen. For example, there were situations where up to four different blues were displayed on the screen. All four colors were very close in terms of LCH. The problem was worse when we took into account the second screen: we had to spread apart further the hue of each blue so as to allow recognition and association within the two screens. However, we did not explore further the problem, as our assignment was only to work on the main radar view. Fortunately, there are other contextual information that allow the controllers to discriminate between the status reflected by the colors. Nevertheless, this problem should not be overlooked.

The temperature of the display also influences perception. For example, we changed the saturation and hue of a slightly colored gray from C=3% to C=2.92% and from H=156° to H=206° to make the values coherent with other colors. We did it offline, and to our surprise, when users saw the new result, they said it was too colored. We learned three things. First, a 50° modification of hue with saturation as low as 3% is noticeable (and hindering). Second, offline modifications are harmful, even if based on sensible reflection made by an experienced graphical designer. Third, this is another example that shows that specifying a gray with R=G=B is harmful, because it does not take into account every parameter that influences color rendering and perception.

*A lasting, iterative activity*

Even though it is possible to roughly describe the workflow we used (design luminosity first, then saturation, then small objects), the actual activity was done in an iterative manner. Besides, as any design activity, the tasks took us some time to accomplish.

First, we had to fix problems introduced by our own new settings: it was difficult to know the impact of a modification, to remember the dependencies between constraints, and to check every possible problem all along the process. Furthermore, we had to explore several configurations, going back and forth between intermediate solutions, which was not an easy task to do with the tools we were using. Besides, designing needs maturation and understanding of the context, for both the designers and the users. For example, designing the right warning orange required the designer to really integrate the conditions of apparition and the context of use of such orange. Regular discussions around the examples and the tools really helped designers and users to achieve a successful result. Finally, designing a color palette is highly subjective. This is not to say that users do not know what they want, but diversity between users, fatigue due to hours of design, changing context conditions etc. make the design subject to unexpected modifications, at best local, at worst global. As designers, we had to react accordingly. For example, in the first task, we worked with users so as to get their feedback and fix problems as soon as possible. After one day of designing, we had a new palette that was satisfying to both the users and the designers. When we came back the following day, the users found that the new configuration made the image too uncomfortable because it was too luminous. We had to lower the luminosity of each color one by one to fix this problem.

## 4 Implication for Design

In this section, we sum up the experience we gained during our tasks. We identify the relevant dimensions to take care of, when designing tools and methods to support graphical elements design.

*Design with actual, controllable examples*
Actual color design tools allow control of color dimensions and checking of the results on a square displaying the resulting color [11]. However, to really design a color, we had to configure the application with the newly designed color, and check it in an actual scene, in our case a radar view. This takes time and prevents an efficient iteration loop. In our ad-hoc tools, we tried to solve this problem by embedding a sample of the flights that were supposed to be organized in flows and sub-flows. This allowed us not only to check the results, but also to completely change the way we handled designing, as we could test multiple solutions quickly, and adjust swiftly and precisely each color. In fact, color-design tools should use an imaging model, not a color model as they do today [6].

*Design with multiple examples at once*
An object may be involved in multiple situations. For example, when designing the color of a flight, we had to take into account all the backgrounds over which it could be displayed. This forced us to go back and forth between different configurations of the application. Thus, a color tool should not only embed controllable examples, but it should also allow an easy switching between examples (either by juxtaposing them, or progressively disclosing them).

*The global scene is important*
We highlighted the importance of designing on real scene samples. However, it is important to keep in mind that these samples are only parts of a global graphical scene. All individual elements build up the perception of the global scene, and global rendering is the only mean to check the global comfort of the UI. Inversely, the global scene influences the perception of a single element. In order to experience these interactions, a designer must work on real scenes, and not just approximate or simplistic ones.

*Foster explorative design*
Making a design successful requires exploring and comparing alternative solutions. Our tools hinder exploration, as they require to save the configuration and to relaunch the application, to compare with early designs. Fortunately, we could use two screens to compare our designs with the configuration currently in use in control centers: this scheme must be generalized to any intermediary configuration, whether it concerns a single element, or a set of elements. Sideviews is an example of such style of design [15].

*Foster constraints expression*
We also noticed the importance of expressing constraints and reifying them. During the design phases, remembering all constraints is difficult. Actually, color molecules implement a kind of constraints, enforced with graphical interactions [10]. Such graphical constraints would have made group settings easier: it would have allowed us to lower

the luminosities of several elements at once. In addition, constraints expressed with formulas would check that a change of a parameter does not violate a previously fulfilled constraint. However it is sometimes difficult to express constraints, either graphically, or even prosaically: the constraints between the sectors gray are complex, and a tool that would enforce them would be too cumbersome to use.

*Expressing and structuring colors*
The LCH model, together with calibrated displays, is the right tool to express color. The LCH color space allows for predictable manipulations and structured design. However, when designing very precise values, the resolution of the machine color model hinders tuning. We were obliged to tune the final RGB values to find the right set of gray level for background. A right tool would facilitate expressing and manipulating the structured relationships between colors while at the same time allowing small adaptations using the final color model.

Even if based on the perceptual system, the LCH model is not perfect. The dimensions are mostly orthogonal, but not perfectly orthogonal. The LCH model does not allow for modifications that would guarantee that a named color is still perceived as the same. Color expression and constraints must take into account the specificities of named colors, and provide suitable interaction to help designers manipulate them.

*Not just about design: integrate all purposes*
During our design activities, we found that our task was not only to reach a final palette, but also to help users express their needs, to help us justify our choices and convince users, and to help accept the new settings. In the justification phase, by giving quantitative arguments, constraints would enable to argument for the choice eventually made. A list of constraints would also act as a proof that criterions required by a specification document are respected, and would help define an experimental plan to experimentally assess the design choices [12].

A tool to help designing should not be used only once, but also as an instrument that would accompany the configured system all along its lifetime. Actually, the tool itself would play the role of the configuration file of the target application. Such a tool would reify the design choices and justifications and help designers understand and respect past constraints that led to a particular design. As such, it would serve as a design rationale tool, and would extend the notion of active design documents [9, 4].

## 5 Conclusion and Perspectives

In this paper, we reported about our experience as designers of colors for graphical elements. We showed that interaction between visual dimensions and display context makes the design very dependent on small details. We reported how we handled various technical, cultural, and perceptual constraints. Based on this experience, we devised a set of implications for designing future instruments to support graphical design activities.

Notwithstanding the specificity of cognitively demanding ATC activities where even the smallest detail is important, the set of implications for design we devised should be of interest in other contexts. For example, web design requires defining a palette, but for a design to be coherent and harmonious, the same concerns that we expressed here should be taken into account. The features of the tool we envision

would be of the same usefulness, whether as a design tool, as a design rationale tool, or as an evaluation tool.

## References


1. Barboni, E., Navarre, D., Palanque, P., Bazalgette, D.: PetShop: A Model-Based Tool for the Formal Modelling and Simulation of Interactive Safety Critical Embedded Systems. In: Proceedings of HCI aero conference, Seatle, USA (2006)
2. Bertin, J.: Sémiologie graphique: Les diagrammes -Les réseaux - Les cartes (Broché) 1070 pages Editeur: Editions de l'Ecole des Hautes Etudes en Sciences janvier 31 (1999)
3. Brewer, C.A.: Color Use Guidelines for Mappingand Visualization. In: MacEachren, A.M., Taylor, D.R.F. (eds.) Visualization in Modern Cartography, ch. 7, pp. 123–147. Elsevier Science, Tarrytown (1994)
4. Boy, G.A.: Active design documents. In: proceedings of the 2nd Conf. on Designing interactive Systems: Processes, Practices, Methods, and Techniques. Amsterdam (1997)
5. Card, S., Mackinlay, J., Shneiderman, B.: Information Visualization Readings in Information Visualization:Using Vision to Think, pp. 1–34. Morgan Kaufman, San Francisco (1998)
6. A Colour Appearance Model for Colour Management Systems: CIE CAM 2002, CIE 159, 2004 (2004)
7. Federal Aviation Administration (FAA). Human factors design standard (HFDS), HF-STD-001 (March 2008), http://hf.tc.faa.gov/hfds
8. Specification ICC.1:2004-10 (Profile version 4.2.0.0) Image technology colour management : Architecture, profile format, and data structure, International Color Consortium (2004)
9. Lacaze, X., Palanque, P., Barboni, E., Navarre, D.: Design Rationale for Increasing Profitability of Interactive Systems Development., Rationale Management in Software Engineering, pp.182-197 (2005)
10. Lyons, P., Moretti, G.: Incorporating Groups into a Mathematical Model of Color Harmony for Generating Color Schemes for Computer Interfaces. In: Proceedings of the 2005 IEEE conference on Virtual Environments, Human-Computer Interfaces, and Measurement Systems, 18-20 July 2005, pp. 80–85 (2005)
11. Lyons, P., Moretti, G.: Nine tools for generating Harmonious Colour Shemes. In: Masoodian, M., Jones, S., Rogers, B. (eds.) APCHI 2004. LNCS, vol. 3101. Springer, Heidelberg (2004)
12. Mackay, E.W., Appert, C., Beaudouin-Lafon, M., Chapuis, O., Du, Y., Fekete, J.D., Guiard, Y.: Touchstone: exploratory design of experiments. In: Conference on Human Factors in Computing Systems, pp. 1425–1434 (2007)
13. NASA Color Usage (2004) (March 2008), http://colorusage.arc.nasa.gov
14. Tabart, G., Athènes, S., Conversy, S., Vinot, J.L., Effets des Paramètres Graphiques sur la Perception Visuelle : Expérimentations sur la Forme, la Surface, l'Orientation des Objets et la Définition des Ecrans. In: IHM 2007 (2007)
15. Terry, M., Mynatt, D.E.: Supporting experimentation with Side-Views. Communications of the ACM 45(45), 1006–1008 (2002)
16. Techniques For Accessibility Evaluation And Repair Tools, W3C workink draft (2000), http://www.w3.org/TR/AERT#color-contrast
17. Ware, C.: Information Visualization: Perception for Design, December 2004, 435 pages, 2nd edn. Morgan Kaufmann, San Francisco (2004)

Info Vis

*Research Paper*

# Visual scanning as a reference framework for interactive representation design

Information Visualization
0(0) 1–16

DOI: 10.1177/1473871611415988
ivi.sagepub.com
SAGE

**Stéphane Conversy[1], Stéphane Chatty[2] and Christophe Hurter[3]**

**Abstract**
When designing a representation, the designer implicitly formulates a sequence of visual tasks required to understand and use the representation effectively. This paper aims at making the sequence of visual tasks explicit in order to help designers elicit their design choices. In particular, we present a set of concepts to systematically analyse what a user must theoretically do to decipher representations. The analysis consists of a decomposition of the activity of scanning into elementary visualization operations. We show how the analysis applies to various existing representations, and how expected benefits can be expressed in terms of elementary operations. The set of elementary operations form the basis of a shared language for representation designers. The decomposition highlights the challenges encountered by a user when deciphering a representation and helps designers to exhibit possible flaws in their design, justify their choices, and compare designs. We also show that interaction with a representation can be considered as facilitation to perform the elementary operations.

**Categories and subject descriptors**
H.5.2 User Interfaces – evaluation/methodology, screen design.

**General terms**
design, human factors

**Keywords**
visualization, infovis, design rationale, visual design, interaction

## Introduction

Designing representation is often considered to be a craft. The design activity requires multiple iterations that mix ad hoc testing, discussions with users, controlled experiments, and personal preferences. These ways of designing are either costly (controlled experiment), error-prone (ad hoc testing), or lead to non-optimal results (personal preference). Although a number of theoretical works help to explain the strengths or weaknesses of a representation,[1–7] no systematic method exists that would help designers to assess their design in an a priori manner, i.e. before user experiments. As suggested by Munzner,[8] such a method would help not only for formative purposes, but also as a summative evaluation before actual user experiments.

When designing a representation, a designer implicitly formulates a way to understand and use the representation effectively. For example, reading a city map requires scanning it, finding noteworthy locations (metro stations, connections, etc.), devising a path to go from one point to another, and so on.[9] For a user, except for very specialized graphics and narrow tasks, figuring out a representation is like interacting using the eyes only: a user has to figure out a solution to his task at hand by scanning the picture, seeking graphics, memorizing things, etc. The succession of these small visualization operations induces a cost that deserves to be evaluated before acceptance of a final design.

We suggest that most design decisions can be explained by the willingness of the designer to reduce the cost of deciphering the representation.

[1]Université de Toulouse, ENAC & IRIT, Toulouse, France.
[2]Université de Toulouse, ENAC, Toulouse, France.
[3]Université de Toulouse, DGAC R/D & ENAC & IRIT, Toulouse, France.

**Corresponding author:**
Stéphane Conversy, Université de Toulouse, ENAC & IRIT, Toulouse, France
Email: stephane.conversy@enac.fr

However, there is no common core of concepts that allows designers to precisely express the rationale behind a design decision. This hinders the design process because it makes it hard for designers to explain to users and stakeholders why a representation is suitable for their tasks (justification) and how a new prototype is better than a previous one (comparison). Furthermore, they cannot justify their choices in a design rationale document, which makes the decisions susceptible to disappearance in future evolutions of the system.

This paper presents a set of concepts for analysing how a user deciphers a representation. It relies on and extends previous works about visual scanning and design elicitation. The goal of the paper is not to show better designs for a particular problem. Rather, the goal of the paper is to present an analysis that exhibits the steps required to figure out a particular representation, and helps to justify design choices and compare representations.

## Related work

We based our work on previous studies that can be roughly divided into three groups. The first group concerns eye gaze, representation scanning, and models of visual perception; the second concerns visual task taxonomies; and the third concerns design formulation.

### *Eye gaze, scanning, and visual perception*

Eye tracking enables researchers to analyse what users look at when solving a problem. However, a large part of the literature is devoted to how to process tracking data in order to analyse it.[10–12] Furthermore, the state of the art in this field still experiments with very low-level designs and abstract graphics,[9,13] far from the richness of today's visualizations. A number of findings are interesting and may help the design of representations, but they are hard to generalize and use in a prescriptive way.[14]

The Adaptive Control of Thought – Rationale (ACT-R) model aims at providing tools that simulate human perception and reasoning.[15] However, the tool is not targeted towards designers, as its purpose is to model human behaviour so as to anticipate real-world usage. It does not take into account some arrangements such as ordered or quantitative layout, nor does it support a description of how a representation is supposed to be used. ACT-R has tentatively been used to carry out autonomous navigation of graphical interface, together with the SegMan perception/action substrate.[16] However, the interfaces used as testbeds are targeted towards WIMP (window, icon, menu, pointing device) applications, which do not exhibit high-level properties available in rich visualization.

UCIE (Understanding Cognitive Information Engineering) is an implemented model of the processes people use to decode information from graphics.[17] Although targeted on graph visualization, UCIE relies on perceptual and cognitive elementary tasks similar to the ones presented here. Given a scene, UCIE can compute a scan path and an estimation of the time needed to get information (with mixed results). However, this work is more targeted at showing the effectiveness of the predictive model than at describing the tasks with enough details to enable designers to analyse their own design and justify it. Furthermore, the tasks do not include operations such as *entering* and *exiting*, or following a path, and their descriptions lack considerations on interaction.

The semiology of graphics is a theory of abstract graphical representation such as maps or bar charts.[4] It describes and explains the perceptual phenomenon and properties underlying the act of reading abstract graphics. In his book, Bertin defines three levels of reading a representation: the elementary level, which enables the reader to 'unpack' visual variables of a single mark, the middle level, which enables the reader to perceive a size-limited pattern or regularity, and the global level, which enables the reader to grasp the representation as a whole, and see at a glance emergent visual information. Bertin[4] pointed out the problem of scanning in what he terms 'figuration' (i.e. bad representation). He briefly depicts how the eye scans a graphic. During scanning, the eye jumps from one mark to the next, while experiencing perturbation by other marks. The eye then focuses on particular marks to gather visual information.

Cleveland has presented a model for studying display methods of statistical graphics.[18] Three visual operations of *pattern perception* are defined: *detection, assembly,* and *estimation*. Three visual operations of *table look-up* are defined – *projection, interpolation,* and *matching*. Similar to our work, the model addresses the method for displaying information. Some operations, such as *projection* (also named *scanning* in the paper) pertain partly to our concerns. However, Cleveland focuses on the efficiency of low-level perception (such as aspect ratio) of statistical information, while we focus on the decomposition in operations for any visualization.

### *Visual task taxonomies*

Casner designed BOZ, a tool that automatically generates an appropriate visualization for a particular task.[19] BOZ takes as input a description of the task to support and relies on a set of inference rules to

generate a visualization that maximizes the use of the human perceptual system. In the following, we use the set of perceptual operators embedded in BOZ, such as 'search (an object with a given graphical property)', 'lookup (a property given an object)', and 'verify (given a property and an object, that this object has the property)'.

Zhou and Feiner[20] designed IMPROVISE (Illustrative Metaphor Production in Reactive Object-orientated visual Environments), another automatic tool to design representations. Zhou and Feiner have refined the visual analysis into multiple levels: visual intents, visual tasks, and visual techniques. Visual tasks include *emphasize, reveal, correlate*, etc. A visual task may accomplish a set of visual intents, such as *search, verify, sum*, or *differentiate*. In turn, a visual intent can be accomplished by a set of visual tasks. A visual task implies a set of visual techniques, such as spatial proximity, visual structure (tables, networks), use of colour, etc.

Previous works pertain to visualization analysis tasks, such as *retrieve value, filter, find extremum, sort*, etc.[21,22] Even if the tasks are presented as low level (compared with higher, more data-related tasks), authors do not discuss how the tasks are 'implemented' in the visualization. Although our framework can be qualified as even lower level, it comes as an addition, because it aims at providing concepts to help discuss the implementation in terms of visual operators.

### Design formulation

The GOMS (goals, operators, methods, and selection rules) Keystroke-Level Model (KLM) helps to compute the time needed to perform an interaction.[23] The Complexity of Interaction Sequences (CIS) model takes into account the context in which the interaction takes place.[15] Both KLM and CIS are based on descriptive models of interaction, which decompose them into elementary operations. They are also predictive models, i.e. they can help compute a measurement of expected effectiveness and enable quantitative comparisons between interaction techniques. These tools have proved to be accurate and efficient when designing new interfaces.[15,23] We relate our work to KLM in the section where we discuss the relationships between visual scanning and interaction.

The speech acts theory,[24] originally aimed at analysing the human discourse, was extended for describing the user's multimodal interaction with a computing system.[25] It provides a successful example of using a model that captures the essence of an interaction modality (speech) and extending it to describe combinations of this modality and others (such as gestures). Our approach to the combination of visualization and interaction can be compared with this.

Green[26] identified cognitive dimensions of notation, which help designers share a common language when discussing design. The dimensions help make explicit what a notation is supposed to improve, or fails to support. Cognitive dimensions are based on activities typical of the use of interactive systems such as *incrementation* or *transcription*. However, they are high-level descriptions and do not detail visualization tasks. Our work has the same means and goals (description and production of a shared language) as cognitive dimensions, but specialized to visualization.

## Idealized scanning of representation

As previously stated, when designing a representation, a designer implicitly formulates a method required to understand and use the representation effectively. The work presented here is an analysis of this method that provides a way to make it explicit.

When trying to solve a problem using a representation, a user completes a visualization task by performing a set of visual and memory operations. A visualization task can be decomposed into a sequence of steps pertaining to the problem at hand (e.g. 'find a bus line'). Each step requires that a sequence of elementary visualization operations be accomplished. Operations include *memorizing* information, *entering* and *exiting* from the representation, *seeking* a subset of marks, *unpacking* a mark and *verifying* a predicate, and *seeking and navigating* among a subset of marks. As we will see below, operations are facilitated by the use of (possibly) adequate visual cues, such as Bertin's[4] selection with colour, size, or alignment. In terms of the model proposed by Munzner[8], we target the *encoding/interaction technique design* box.

In the following, we analyse *idealized* scanning of representations. We use '*idealized*' in the sense that the user knows exactly what she is looking for, knows how to use the representation so as to step through with the minimum necessary steps, and uses only the available information in the representation otherwise stated. Thus, we do not take into account other phenomena such as learning, understanding, error, chance, or personal perceptual disabilities (such as colour blindness). This is similar to the approach taken with the KLM: when applying a decomposition, the designer analyses an idealized interaction.

In fact, the model enables either comparing multiple scanning strategies for a given task and a given representation or comparing multiple representations for a given task and the most efficient scanning. In the following, we focus on representation comparison, and we assume that we have found the most efficient

scanning for each representation. The next section uses an example to illustrate how to perform an analysis of representation scanning. Based on this, we further detail the steps and operations required, and what factors affect users' efficiency at achieving them.

## A first glimpse: A tabular bus schedule representation

There is no such thing as an absolutely effective representation; to be effective, a representation must minimize the amount of work required to fulfil a task.[19] In the following example, the problem to be solved by a user is to answer the following question: 'I am at the IUT Rangueil station and it is 14:18. How long will I have to wait for the next bus to the Université Paul Sabatier station?' The user knows that two bus lines go to the destination (Nos. 68 and 108). Figure 1 is an excerpt of a typical representation of a bus schedule. The display is a physical panel at the station booth, on which lies paper sheets, each with a table for one bus line that displays the time of departure from each station.

The drawings overlaid on the representation show the idealized visualization tasks a user must perform when trying to answer the question. A circle depicts an eye reading, an arrow an eye movement. Memory operations are depicted with a blue 'M'. The step numbers are in the form xyz, which means that step y is the yth substep of step x, and step z is the zth substep of step y. A check mark depicts the last operation of a substep, together with a green circle. Figure 1 also shows two different scanning strategies to answer two instances of an intermediate question ('when is the next bus for line 68 (resp. 108)?').

- Step 0: The user should memorize the two compatible bus line numbers and the current time.
- Step 1.x: The user should find an appropriate bus line. The number of the line is represented in large boldface text at the top-right corner of each paper sheet.
- Step 1.2.1: The user should find his current location ('IUT Rangueil') among the list of stations. The list is a subset of marks of kind 'text', aligned vertically, with no marks in between. The stations are ordered according to their location along the bus line.
- Step 1.2.1.x: The user must find the next departure time. He has to navigate through a row of texts that displays the hour and minute for each bus departure. As the x dimension is multiplexed (or 'folded on') y, the user may not find a compatible time in the first row examined: in this case he has to start step 1.2 over by moving to the next row (Step 1.2.2). Finally, the user finds the next departure when he identifies the first departure that is later than the current time.
- Step 1.3.x.x: The user tries a different scanning strategy than the linear approach above, where only the first time in the row is checked, until a time later than the target is found. The user then performs a back step to the previous row (Step 1.3.3.2). This strategy is often faster, but requires one more memory operation to memorize the previous mark position.
- Step 1.3.3.2: The user finds that this row does not contain relevant information, so he performs a back step to the previous row. This requires memorization of a previous mark position.
- Step 1.2.2.x, 1.3.2.x, green circle: The user finds a compatible bus in each line and thus has to perform mental computation (a difference between two times) to find the duration before the next bus, and memorize it to compare with previous or subsequent findings.

## Elementary operations

This section details the various elementary operations required to implement the steps. To define the set of operations, we analysed a number of different visualizations from multiple domains (various ATC (air traffic control) systems, time scheduling such as tables, clocks or calendars, widgets, visual languages, etc.) We also based our analysis on existing literature when available, supplemented by interviews with visualization designers. Together, the set of operations we came up with is sufficient to explain most design choices, but further, more systematic work may be needed to make the set more comprehensive.

For each operation, we detail it and give elements that aid or hinder operation achievement. We also compare our operations to the BOZ and IMPROVISE taxonomies and explain the differences, mainly in terms of elements that may aid or hinder the operation.

### *Memorizing information*

To solve problems, users have to know what information to seek. They have to memorize this information, so as to compare it with the information that arises from the representation. As we will see in the examples, different representations require different numbers of memory 'cells'. For example, in the tabular bus scheduling view, users need three cells of information at the beginning (current time, 68 and 108), two

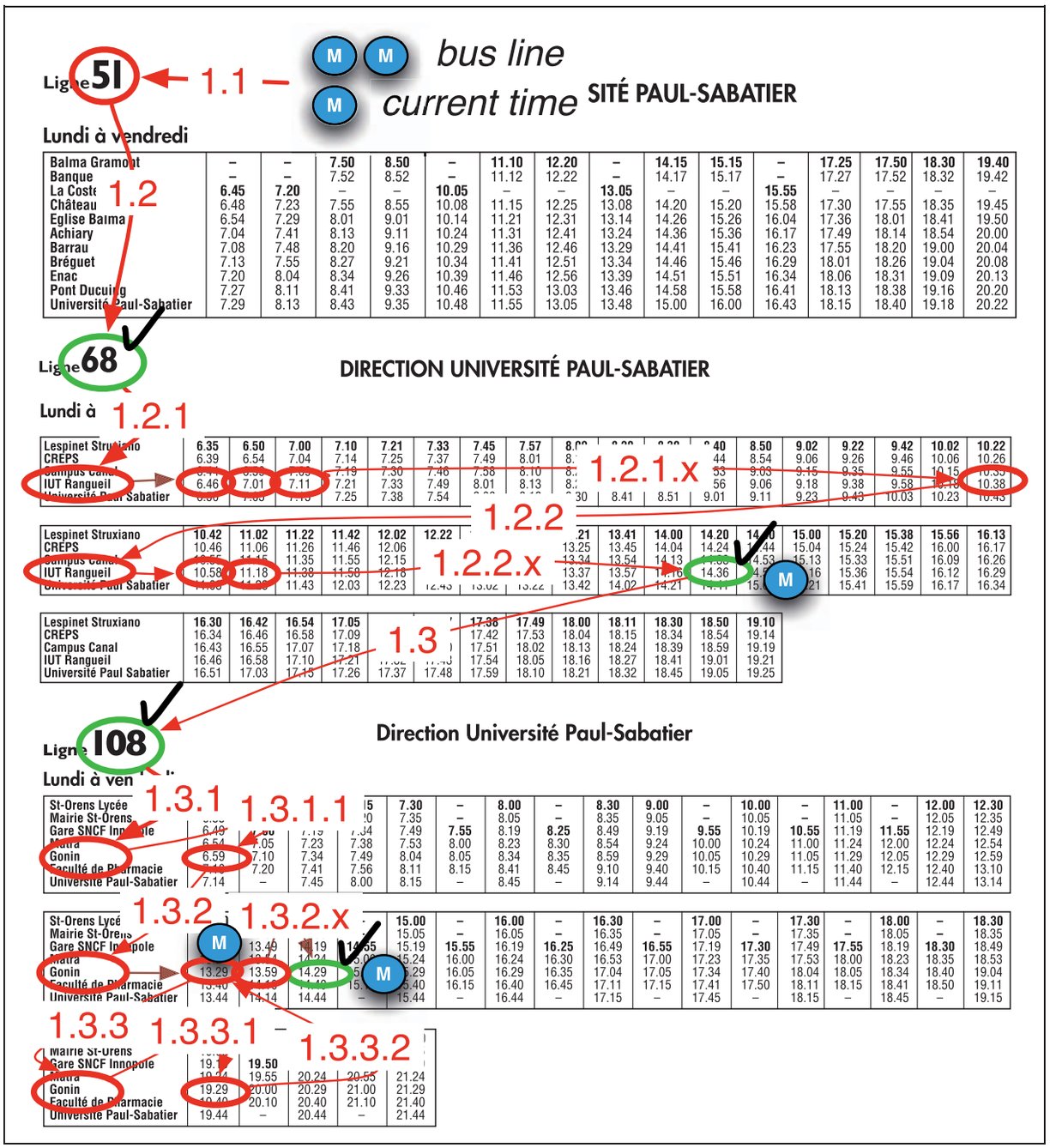



**Figure 1.** A bus schedule representation with the required steps to find particular information.

cells for intermediary results, and one cell for a previous location. Memory requirements are often overlooked when comparing visualizations: the more cells required, the harder it becomes to solve a problem. Memory fades with time, so for long scanning tasks users may have forgotten important information before the end of the scanning. Forgotten information that is available on the representation can be compensated for by additional seeking operations or by adding the equivalent of a selectable visual property (e.g. a hand-written mark or a pointing finger).

## *Entering and exiting representation*

A representation is rarely used in isolation. Users are surrounded by different representation from various

systems. For example, air traffic controllers (ATCos) employ radar views, various lists of flights, paper strips, etc. When they solve a problem, users may have to switch representations. This may require translating the input of a representation into the visual language of another representation and translating the information found back into the problem.

In the bus schedule example, users may have to translate the representation of a time seen on a watch into numbers in the form hh:mm so as to comply with the ordered-by-time menu-like vertical representation (*entering*). They also have to get the correct bus line somewhere (i.e. a map representing the public transportation network) and translate the information (a textual number or a colour) into the visual language of the representation (*entering*). The information to find is the waiting time for the next bus. The tabular representation does not give this information directly and thus requires a mental computation (*exiting*). In the city map example, translating map direction to real-world direction and recognizing street layout is easier if the map is orientated to the terrain (i.e. north of map matching the actual north direction). Taking into account this step is important when a switch of representation does not require translation, as this makes the second representation easier to understand.

## Seeking a subset of marks

When users search for bus line information, they have to search for a subset of the marks in the representation. In order to find the correct line, the user has to navigate from line number to line number.

Perceiving a subset is made easier with *selective* (in the sense of the semiology of graphics[4]) visual variables: marks can be extracted from the soup of all marks at one glance, which narrows down the number of marks to consider. For example, the number of the bus line is represented in text, with a large font size and boldface, placed at the top-right corner of the sheet. The size and position of bus line number make the marks *selectable*. Furthermore, when elements in a subset are close enough together, no other in-between element perturbs the navigation from mark to mark. The list is even easier to navigate in, since the marks are aligned horizontally and vertically (or in other words, marks differ by only one dimension (x or y)).

Conversely, perceiving a subset can be harder in presence of similar marks that do not belong to the considered subset. In the tabular schedule example, all time information has similar visual properties except for the start time of each bus, which is set in bold. If the start time were set in regular, it would be harder to find at a glance. Seeking a subset corresponds to the *search-object-*$\star$ perceptual operator in BOZ.[19]

## Unpacking a mark and verifying a predicate

When the user sees a candidate mark, he or she has to assess it against a predicate. In the tabular bus schedule example, the user has to find a line number that matches one of the correct buses. Assessing a predicate may require extracting (or unpacking[4]) visual dimensions from a mark. This is what Bertin calls 'elementary reading'.[4] This operation also corresponds to the *lookup-*$\star$ and computation perceptual operator class in BOZ.[19] However, assessing a predicate may also require cognitive comparison to memorized information ('Is the bus number I'm looking at one of the memorized ones?') or visual comparison with another mark (example in the following). In BOZ the difficulty of accomplishing the operation depends on the visual variable used, but not on other considerations such as memory or visual comparison.

## Seeking and navigating among a subset of marks

Within an identified subset, a user may search for a particular mark. If marks are displayed in random positions, finding a mark requires a linear, one-by-one scanning of marks, with a predicate verification for each. The time needed is $O(n)$. If marks are ordered (as in the ordered-by-time schedule), a user can benefit from this regularity to speed up navigation, for example by using a binary search approach, which leads to a time needed of $O(\log(n))$. If marks are displayed at quantitative positions, we can hope to achieve $O(1)$. However, this may require secondary marks such as a scale ticks and legends. In this case, scanning is split into two phases: navigating into the scale first, then into primary marks.

Navigating inside a list of texts is equivalent to reading a menu, for which performance may be predicted quite accurately.[27] However, some graphical elements may hinder navigation. For example, navigating in a row surrounded by other rows, as in a table, is difficult. This is the equivalent of a visual steering task:[28] it requires that the eye be able to stay in a tunnel. Some representations are supposed to aid this (e.g. think of a spreadsheet where every other row is coloured). Performance depends on the width and the length of the tunnel. Navigating inside a vertical list of text is easier than navigating in a horizontal one, as a horizontal row is as narrow as the height of a glyph. Furthermore, in particular cases, navigating may require a *step back* to a previous mark, which in turn

XML Template (2011)
K:/IVI/IVI 415988.3d
[15.7.2011–2:26pm]
(IVI)
[1–16]
[PREPRINTER stage]



requires memorizing a previous location (see step 1.3.2.x in Figure 1).

No BOZ perceptual operator corresponds to this operation. IMPROVISE generates scales for quantitative data, but no mechanism facilitates ordered data. None of the taxonomies in BOZ and IMPROVISE handle navigation or take visual steering into consideration.

## Formulating design rationale

We argue that a designer implicitly designs a required sequence of elementary operations when inventing a new representation. We also suggest that most explanations given by designers can be expressed in terms of elementary operations, and in particular in how a particular design improves operation performance. In the following, we present various designs for bus schedules and ATC paper strips. We explain the expected gains of each design using the concepts presented above. We balance the claims by our own analysis and possible loss of performance due to a lack of support for overlooked operations.

### *Bus schedule*

*Ordered-by-time linear representation.* One bus company proposes the representation in Figure 2 on its website. This displays an ordered list of time of departure at the chosen station along the x dimension, with the corresponding bus line indicated by a cell containing a background colour and white text. The required steps are:

- Step 0: Memorize the current time and appropriate bus lines (entering and memorizing), possibly translating time from an 'analogue' watch to a text in the form hh:mm (entering).
- Step 1: Find the ordered list of time (seeking) and the first time later than the current time (navigating and predicate).
- Step 2: Find the next appropriate bus (predicate, or seeking a mark if using bus colour).
- Step 3: Find the associated time (seeking a mark).
- Step 4: Compute the waiting time before the departure (exiting).

Compared with the tabular representation, the following operations may be aided. . .:

- Seeking and navigating among a subset of marks: times of departure are displayed in a ordered manner which may ease navigation.
- Seeking a subset of marks: the user can easily select elements to the right of the element found in step 2 (later times, using selection based on location).
- Memorizing: there is less information to memorize (two vs. six chunks). . ..and there are no apparent drawbacks.

*Spiral representation.* SpiraClock is an interactive tool that displays nearby events inside a spiral (instead of a circle, like a regular clock).[29] Time of event is mapped to angle, and thanks to the multiplexing of the angle over the radius, other information emerges (periodicity, closeness through radius) (Figure 3, left). The clock also displays the current time and adapts the event occurrences accordingly. The occurrence of an event is actually depicted by the 'most recent' limit of a 'slice'. Duration is a relative angle, or a curvilinear distance, which is quantitative representation, more precise on the exterior of the spiral (i.e. for close events) than in the interior. There is also a scale depicted with black squares along the circle. SpiraClock's designers argued that adding textual information about hours would be useless, since the design uses a well-known reference (a watch) and since the visualization is focused on current time. If we represent the bus timetable on SpiraClock (as in Figure 3, left), the steps required to answer the question are:

- Step 0: Memorize two bus colours (*entering* and *memorizing*).

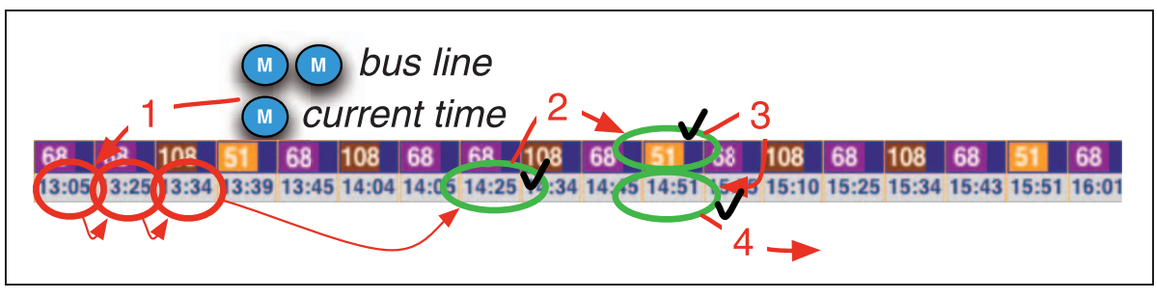



**Figure 2.** An ordered-by-time bus schedule.

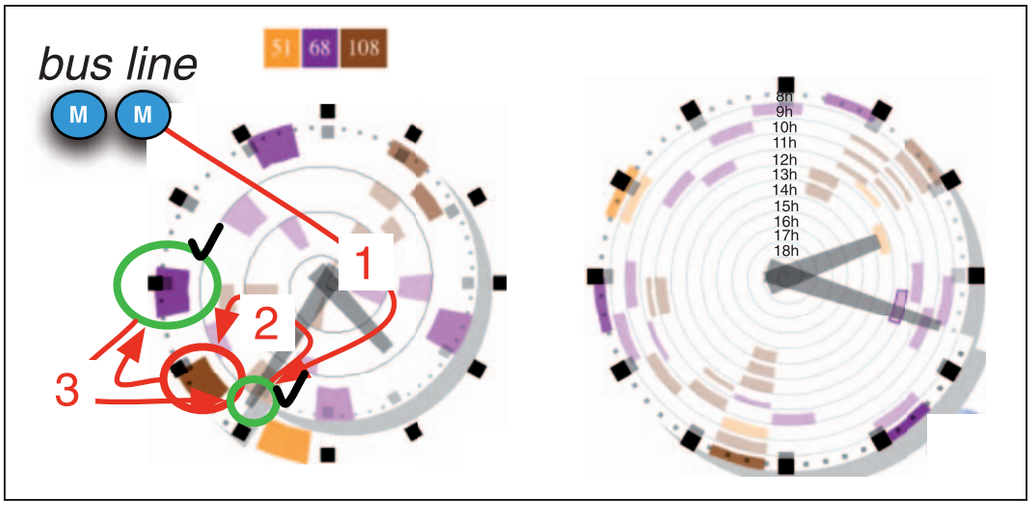



**Figure 3.** SpiraClock. Left: visual scanning. Right: a configuration that displays more information.

- Step 1: Find the end of minute hand (*seeking a mark*).
- Step 2: Find the next matching coloured mark (i.e. corresponding to line 68 or 108) (*seeking a mark*).
- Step 3: Evaluate the distance between the matching mark and the minute hand, and estimate the waiting time (*unpack* and *exiting*).

Compared with the ordered linear representation, the following operations may be aided. . .:

*Entering:* The current time is directly visible thanks to the hands.

*Navigating*: Since the time is visible, navigating to the next correct bus is shorter.

*Exiting*: With SpiraClock, a rough idea of the waiting time is directly visible (no computation needed), as it is proportional to distance and the design uses a culturally known scale. . . . and there are no apparent drawbacks.

*Quantitative linear representation.* Figure 4 shows a representation based on a linear quantitative scale. Each coloured rectangle represents the departure of a bus at the chosen station. The horizontal position of a rectangle corresponds to the time of departure and is multiplexed along the vertical dimension. To aid navigation, a linear scale is provided, with textual information about hours and small ticks to mark quarters between hours.

- Step 0: Memorize two bus colours (*memorizing*), possibly translate time from a watch to a text for hour, and then to a position among ticks for minutes (*entering*).
- Step 1: Find the hour (seeking a mark).
- Step 2: Find the correct quarter-hour among the ticks (seeking a mark).
- Step 3: Find the next compatible bus (i.e. corresponding to line 68 or 108) (seeking a mark).
- Step 4: Evaluate the distance between the matching mark and the minute hand, and estimate the waiting time (no computation is needed) (*unpack* and *exiting*).

Compared with SpiraClock, the following operation may be aided. . .:

*Navigating:* Thanks to the linear layout and the supplemental space between rows, the steering task is easier to perform (especially comparing the narrow tunnel configuration of Figure 3, right). . . . at the expense of the *entering* operation (there is no current time visible since the representation is not dynamic).

The equivalent radial visualization is shown in Figure 3, right: the amount of information is the same (all events in a day are displayed), and a scale of hours helps readers *navigating* into the visualization. Discussions about both designs can revolve around the ease with which a viewer can navigate in a narrow tunnel, be it linear or spiral.

## ATC strips

The activity of ATCos includes maintaining a safe distance between aircraft by giving clearances to pilots – heading, speed, and level (altitude) orders. ATCos must detect potential conflicts in advance. To do this they use various tools, including a radar view and flight strips.[30] A flight strip is a paper strip that shows the route followed by an aeroplane when flying in a sector (Figure 5).

XML Template (2011)
K:/IVI/IVI 415988.3d
[15.7.2011-2:26pm]
(IVI)
[1–16]
[PREPRINTER stage]



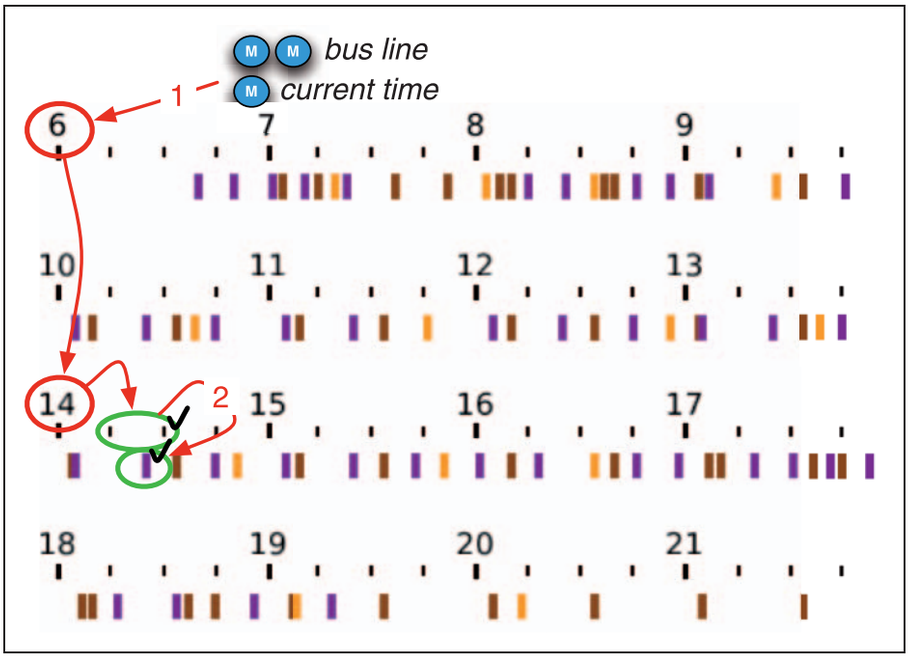



**Figure 4.** A linear, quantitative bus schedule representation.

The route is presented as an ordered sequence of cells, each cell corresponding to a beacon, with its name, and its time of passage. Controllers lay paper strips on a strip board, usually by organizing them in columns. The layout of strips on a board, although physical, can be considered as a representation. Some planned systems aim replacing paper strips with entirely digital systems so as to capture clearances in the database (currently the system is not aware of clearances from the controllers to the pilots). These systems partly replicate the existing representation and we show in subsequent sections how they compare with respect to representation scanning.

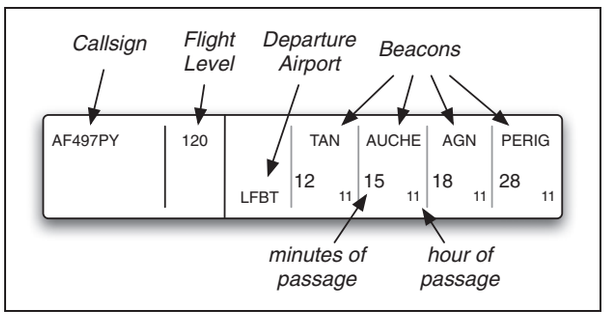



**Figure 5.** An ATC paper strip.

*Regular strip board.* One of the activities of a controller is to integrate the arrival of a flight into the current traffic. To do this safely, the controller must check that for each beacon crossed by the new flight, no other flights cross that beacon at the same time at the same level. Figure 6 shows the required idealized scanning, with typical paper strips organized in a column. The steps are:

- Step 1: Find the flight level and memorize it (*seeking* and *memorizing*).
- Step 2.1: Find the beacon text on the arrival strip (*seeking*), and for each beacon (horizontal text list scanning, with no perturbation), do the following steps (*navigating*).
- Step 2.2: Memorize the beacon text, find the minute information (hour is usually not important) (*seeking*) and memorize it (*memorizing*).
- Step 2.3: For each other strip (vertical rectangular shape list scanning), do the following steps (*seeking* and *navigating*).
- Step 2.4.1: Find the beacon text, and for each beacon (horizontal text list scanning, with no perturbation), do the following steps (*seeking* and *navigating*).
- Step 2.4.2: Compare the beacon text to the one memorized in step 2.2 (*predicate*).
- Step 2.4.3.1: If it is the same, find the minute text, and compare it to the one memorized in step 1.2 ($\pm$ 5 min) (*predicate*).

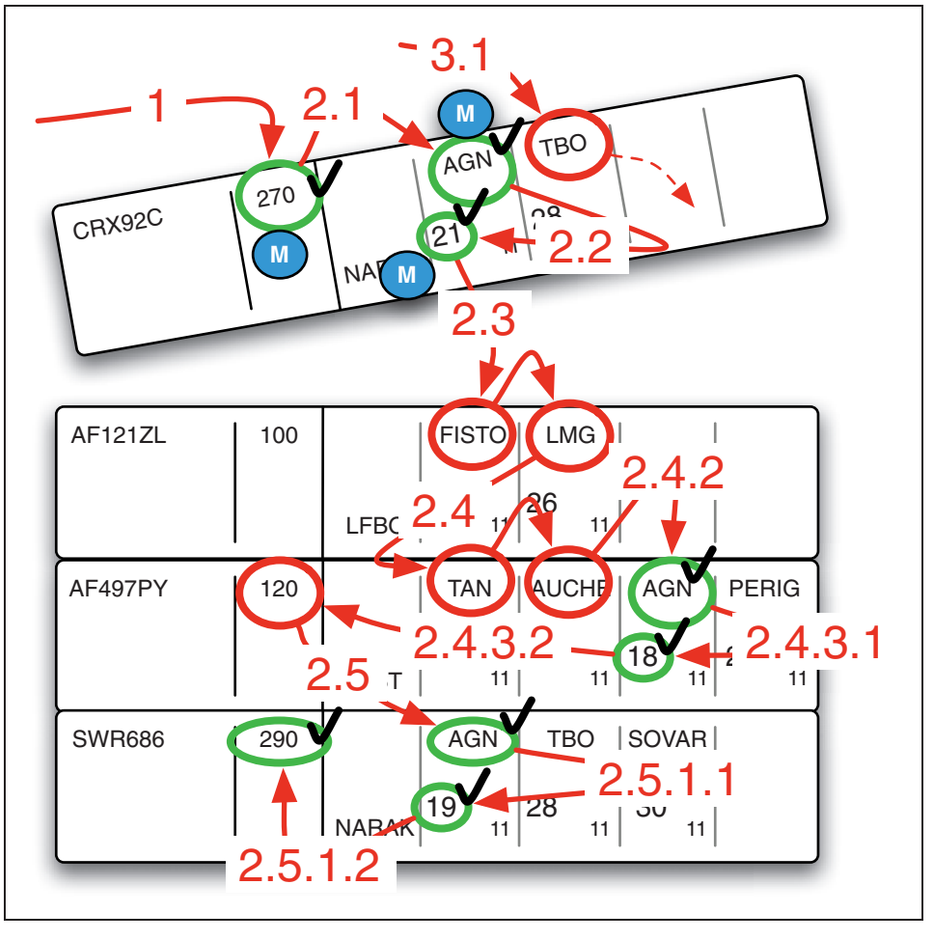



**Figure 6.** Scanning on regular ATC paper strips.

- Step 2.4.3.2: If the number is about the same, find flight level, check it and compare it with the memorized level (*predicate*).
- Step 2.5.1.2: If it is the same, do something to avoid a conflict (*predicate* and *exiting*).

*Strips in coloured holders.* The strip look and layout in the previous section is specific to the en route control centre in Bordeaux, France. In other en route control centres, people use rigid, coloured holders for each paper strip. The look of the strips is different, as the coloured frame of the holder surrounds each strip. Figure 7 shows an idealized scanning with coloured strip holders: here red is for north–south flights (odd flight level), while green is for south–north flights (even flight level). Because of the different level assignments, controllers can be sure that red and green flights will never enter into conflict. Red holders can quickly be extracted from green ones (selection based on colour). Hence, coloured strip holders enable controllers to narrow the set of flights to compare with a new one, and reduce the number of required steps accordingly (step 2.x, with $x \geq 3$, *seeking and navigating*). Holder colours can also ease *predicate verification*: holder colour of the arriving strip can be matched easily to holder colour of other strips, without requiring the controller to determine if the strip is a north–south or a south–north flight.

*Dynastrip.* Dynastrip displays beacons in a quantitative way, mapping time to the horizontal dimension (Figure 8).[31] All time scales are aligned across strips. The main goal of Dynastrip is to display the position relative to the planned route in the strip, which adds information. Dynastrip designers also hoped that this representation would assist controllers to identify conflicts: if beacons with the same text are vertically aligned, it means that multiple flights pass over the same beacon at the same time.

- Step 1: Find the flight level (*seeking*) and memorize it (*memorizing*).

XML Template (2011)
K:/IVI/IVI 415988.3d
[15.7.2011–2:26pm]
(IVI)
[1–16]
[PREPRINTER stage]



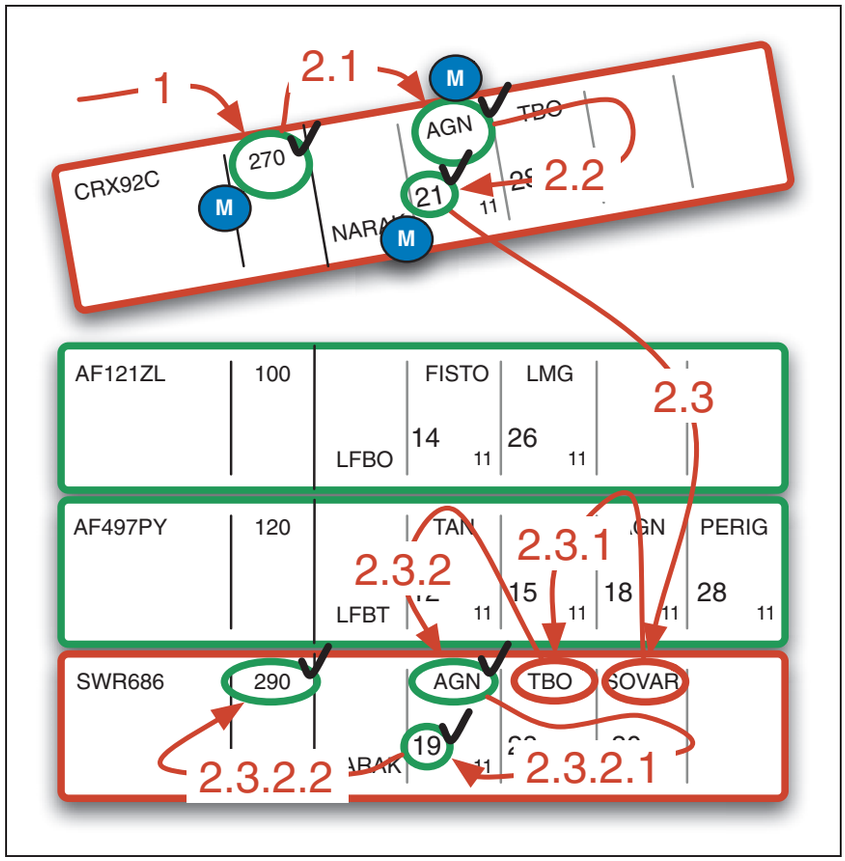



**Figure 7.** Scanning with paper strips in coloured holders.

- Step 2.1: Find the beacon texts on the arrival strip, and for each beacon (horizontal text list scanning (*seeking* and *navigating*), do the following steps.
- Step 2.2: Memorize the beacon (*memorizing*), steer visually through a tunnel (± 5 min) (symbolized in grey on Figure 8 but not shown on the actual interface) (*seeking* and *navigating*), and compare each beacon found with the memorized one (*predicate*).
- Step 2.2.1: Find the flight level, check it and compare it with the memorized level (*predicate*).

Compared with the regular strip boards, this design may aid. . .:

- *Seeking and navigating*: thanks to a steering task, beacon search is facilitated.
- *Verifying a predicate*: the time limit is directly visible. . . . at the expense of a supplemental interaction to reach beacons not yet visible on the time scale.

## Validity and limitations

Idealized scanning is only theoretical. We have not verified experimentally the degree to which actual scanning corresponds to our model, which raises questions about the validity of the work presented here. However, we suggest that designers rely implicitly on idealized scanning, although their expectations do not always stand against reality.[32] A deeper understanding of the phenomena is thus necessary to make design choices and expected benefits (explicit), and to get a reasonable confidence in the design.

Bertin's semiology of graphics and Furnas' Effective View Navigation[33] have not been fully validated experimentally. Nevertheless, their concepts permeate a large number of visualization designs. These approaches allow identification of relevant concepts and dimensions when analysing or designing new visualizations. We think that the elementary operations we identify in this paper will serve as a similar framework for representation rationale. In the same way, we have not verified experimentally whether navigation in an ordered set is easier than in a random set, and whether navigation in a quantitative set is easier than in an ordered set. Again, a number of visualizations rely on these assumptions: making the assumptions explicit helps designers think about the effectiveness of their designs.

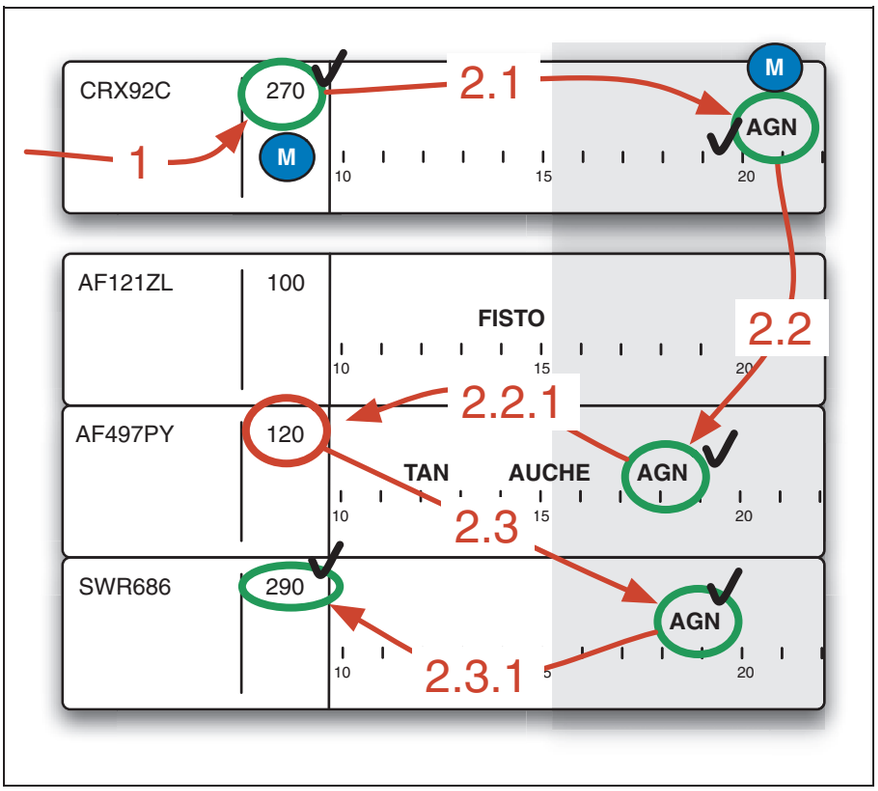



**Figure 8.** Dynastrip, overlaid with the steering tunnel.

The absence of a distinction between 'beginners' and 'experts' in our analysis seems problematic as well. This is clearly the case in the ATC example: we know from previous observation that ATCos do not scan the strips in the same way as we described the process above. Instead, they rely heavily on their knowledge of the sector, recurrent problems, and recurrent aircraft to detect conflicts. Again, our description aimed at eliciting what the visualization enables for a reader that only uses information extracted from the representation. However, during normal operations, ATCos regularly do what they call a 'tour of the radar image' or a 'tour of the strip board' in order to check 'everything'. In this case, they are supposed to scan heavily both representations and they may exhibit some of the theorized behaviour. Furthermore, we observed that ATCos make more errors when training on a new sector, at least partly because of representation flaws. These flaws are compensated for by expertise, which is somewhat related to knowledge in the head and memory (in some cases, ATCos are considered as experts on a sector only after 2 years of training). However, in high-load situations, with many aircraft, or with particular problematic conditions such as unexpected storms, the representation becomes more important and controllers seem more likely to exhibit the theorized behaviour.

## Visual scanning and interaction

Very few serious visualization applications are devoid of any interaction with the user, whether for saving data, searching, modifying data, or changing the representation itself. Even bus schedules printed on paper are often bound in leaflets that the user must browse to find the appropriate page. Zooming and panning, changes of view point, data filtering, and similar operations are often involved to help the user navigate in the data representation. Considering visual scanning as interaction that occurs through the eyes, this can be understood from two equivalent theoretical points of view: actions as part of reading a representation, or visual scanning as part of interaction in general. Or, from a more practical perspective, it can be considered as the choice of a new representation by the user.

In this section, we first explore the more practical perspective and use examples to demonstrate how the user, by interacting with the representation, plays a similar role to the designer: she selects a new representation that makes visual scanning simpler for the task at hand. We then discuss the more abstract perspectives, outlining how these users' actions could be described in the same framework as the visual scanning itself, thus allowing designers to reason about

XML Template (2011)
K:/IVI/IVI 415988.3d
[15.7.2011-2:26pm]
(IVI)
[1–16]
[PREPRINTER stage]



how their overall design will be used and not only the individual representations.

## *Interacting for better representations*

*Pen-based digital stripping system.* Figure 9 shows a digital pen-based system that adds an interaction which allows the controller to press a beacon cell so as to highlight in red the time of passage over that beacon on other strips (the system cannot automatically detect conflict because the data on the strips are not always current). This facilitates *seeking and navigating* in step 2.x as it reduces the subset of marks to consider when comparing times and *memorizing* (one vs. three cells).

*Progressive disclosure.* Progressive disclosure dictates that detail be hidden from users until they ask or need to see it, in order to avoid overwhelming users with information.[34] Progressive disclosure is often implemented with simple property boxes, on which properties can be expanded (using a 'show more' button, or a 'disclosure triangle' in Mac OSX toolbox). As such, this design principle can be considered as a way to ease *navigation* between important elements, before explicitly hardening it when navigating has been achieved successfully.

*Switching views.* Calendar systems (such as Apple iCal or Google Agenda) often offer multiple views on events information. In a month view, events are ordered on the y screen dimension, whereas in a week view, events are displayed in a quantitative manner on the y screen dimension. Switching from month to week view enables users to *unpack* the duration information of events more easily. Switching from week to month view enables users to visualize more events (the month view is denser) in an ordered manner, and thus facilitate *navigating*.

*Brushing and selection.* Brushing enables users to select a subset of displayed data in a visualization system. The feedback of such an interaction usually highlights the brushed data, by changing their colour for example. Brushing in a matrix scatterplot can be used to detect patterns in other juxtaposed

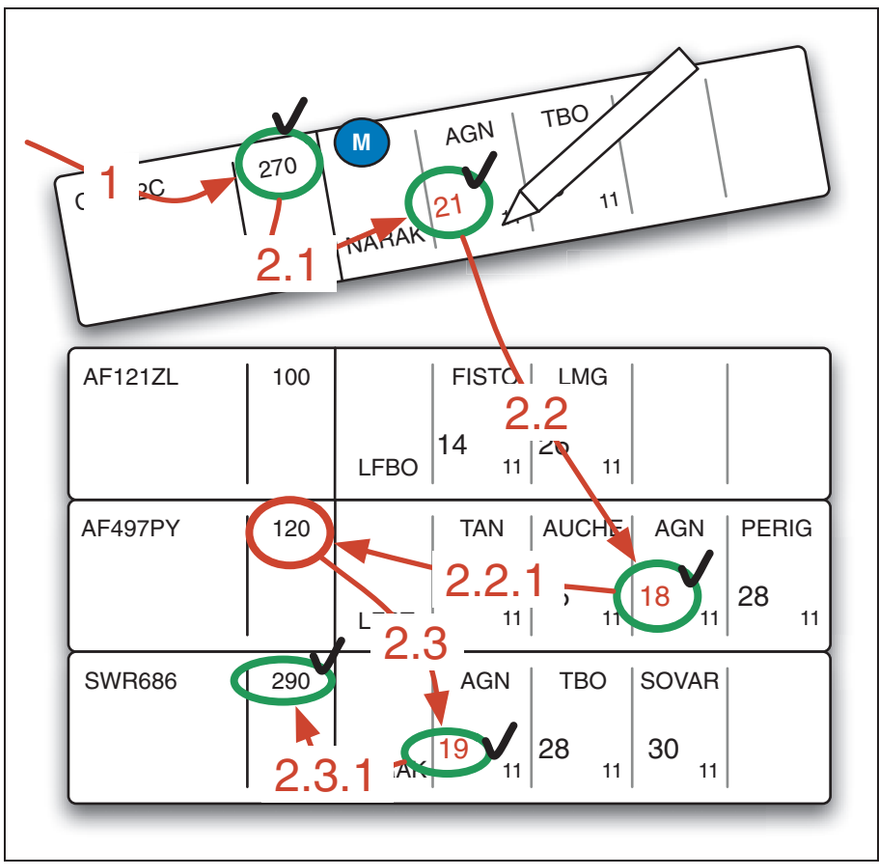



**Figure 9.** A pen-based digital stripping system that enables highlighting of information.

scatterplots, but it also can be used to find a particular plot in other juxtaposed views. The last case can be considered as a way to facilitate *exiting* and *entering* between two juxtaposed views.

*Progressive transition between views.* With calendar systems, switching makes a new view replace the current one: views are at the same place, converse to juxtaposed views. The switch is instantaneous, which disturbs the optical flow of users. Hence, users are forced to scan the new visualization to find again the particular information they were looking at in the previous view. Thus, to perform a switch of views, users have to *exit* the first view by *unpacking* and *memorizing* conceptual information (day, hour) and *entering* the new visualization.

On the other hand, a progressive transition between views enables users to track moving marks during the time of the transition. For example, ScatterDice[35] use an animated three-dimensional (3D) rotation between ScatterPlots. Progressive two-dimensional (2D) interpolations also provide transition between scenes.[36,37] Both transitions (2D or 3D) enable users to track a particular moving item and see its final position in the final view. Hence, progressive, animated transitions enable users to get rid of *exiting* and *entering* views that occupy the same place. Moreover, tracking a moving mark is like guiding the eye of the user, by controlling it (in the sense of control engineering). The goal is the same with a spreadsheet where every other row is coloured.

## Discussion

The concept of *interacting to perceive better* is not new: in fact, this is a concept that is shared among psychologists of action and perception.[38] Designers adapt the representation to make it easier to answer specific preestablished problems. Users also adapt the representation to make it easier to answer a problem at hand. Hence, interacting to change views is of the same nature as designing. In both cases, the present work is helpful as an account of the visual task at play, but it is not helpful at describing the 'design manoeuvre'[26] required to get a better design. There may be new concepts remaining to be identified, in both the design space and the user space, which would form the basis of a prescriptive method for designing better (interactive) representations.

Designing an interactive representation cannot be as simple as taking into account visual scanning alone, nor can it be as simple as counting the number of KLM operators alone. The design must be analysed as a whole, and actions to switch from a representation to another should be taken into account. Interestingly, KLM, despite being focused on the users' actions, accounts for its perception and memorization activities through its operator M. One could consider our work on visual scanning as a first attempt at describing some aspects of this operator in more detail. One can use this perspective to extend our approach to representations that the user can manipulate, indifferently considering actions as part of the scanning process or scanning as part of a global interaction process. One way of proceeding would be to add an interaction operation to the visual scanning language. This would provide user interface designers with two dual languages for analysing their designs: one focused on the user's physical actions, with operator M used to capture other types of interaction, and the other focused on visual scanning, with operator I used to capture other types of interaction. At a finer level of analysis, the two languages would then appear to be simplified and practical versions of a common language that describe all interaction operations on the same foot.

Note that considering scanning as interaction is not as artificial as it may seem. On the one hand, at the physical level there is some interaction through the emitted light, and it does indeed trigger significant changes in the user. And on the other hand, the use of speech that acts to describe multimodal interaction has shown that combining different interaction modalities in the same abstract framework can provide designers with an adequate description language. Finally, proponents of enaction think that perceiving is acting: 'the content of perception is not like the content of a picture; the world is not given to consciousness all at once but is gained gradually by active inquiry and exploration'.[39] If this theory proves true, the total costs of adapting the view and scanning would be difficult to estimate with a method as simple as summing the cost of individual operations. A finer language that accounts for the concurrency between operations might prove more suitable with this regard.

## Conclusion

In this paper, we presented a method to analyse idealized scanning of graphical representations. The method relies on a set of elementary operations, which includes operations from previous taxonomies and new ones (*entering*, *exiting*, *memorizing*) together with new considerations (back steps, visual steering, and the use of ordered or quantitative arrangement). We argue that rationale for design can be expressed in terms of these elementary operations. We showed in various examples how such an analysis can be achieved and how gains and losses can be explained with

elementary operations, including when considering interaction as a change of representation. The set of elementary operations forms the basis of a shared common language that helps designers justify and compare their choices.

In its current form, the method is descriptive, not predictive. We believe that we are still far from a fully predictive model of human performance in representation use, if only because performance depends on multiple external factors, as demonstrated by Lohse.[17] Instead, we take another perspective: we argue that a descriptive-only method is useful for designers, since the decomposition highlights the challenges encountered by a user when deciphering a representation. The benefit is equivalent to one of the two benefits of KLM: in addition to predicting completion times, KLM helps designers to understand what a user must do to accomplish an interaction task. Similarly to KLM, this makes our model a comparative one, as it helps designers to choose designs based on a sound analysis.

In addition to the examples presented here, we have successfully applied our analysis method presented to other representations, such as item rating by customers in online stores, widgets, and radar images. Work is certainly needed to expand the set of operations and the elements that aid or affect their realization. For example, we do not yet take into account the fact that tasks can be aided when externalizing constraints into the real world,[40] nor did we take into account representations that ease mental computation.[41] Furthermore, different acts of mental computation and memorization may exhibit very different costs. In addition, while we tackled the 'what to do' question in this paper, we did not tackle the question of 'how to do it'. Eventually, we need to propose a systematic method that will help designers find for themselves the steps and considerations to take into account when evaluating the effectiveness of a particular representation.

## Acknowledgements

The authors wish to thank the reviewers for their very relevant suggestions. We also thank N Roussel and WG Philips for their helpful comments.

## References

1. Amar R and Stasko J. A knowledge task-based framework for design and evaluation of information visualizations. *Proceedings of IEEE InfoVis '04*. IEEE Computer Society, Washington DC, 2004, pp.143–149.
2. Anderson JR, Matessa M and Lebiere C. ACT-R: A theory of higher-level cognition and its relation to visual attention. *Hum Comput Interact* 1997; 12: 439–462.
3. Anderson JR, Bothell D, Byrne MD, Douglass S, Lebiere C and Qin Y. An integrated theory of the mind. *Psychol Rev* 2004; 111: 1036–1060.
4. Bertin J. *Sémiologie Graphique – Les diagrammes – les réseaux – les cartes*. Paris: Gauthier-Villars et Mouton & Cie. Réédition de 1997, EHESS, 1967.
5. Cleveland W and McGill R. Graphical perception: Theory experimentation and application to the development of graphical methods. *J Am Stat Assoc* 1984; 79: 531–554.
6. Tufte ER. *The Visual Display of Quantitative Information*, 2nd ed. Graphics Press: Cheshire, CT, 2001.
7. Wilkinson L. *The Grammar of Graphics*. Springer Verlag: New York, 1999.
8. Munzner T. A nested process model for visualization design and validation. *IEEE Trans Vis Comput Graph* 2009; 15: 921–928.
9. Ware C. *Visual Thinking for Design*. Morgan Kaufmann, Waltham, MA, 2008.
10. Crowe EC and Narayanan NH. Comparing interfaces based on what users watch and do. *Proceedings of the 2000 Symposium on Eye Tracking Research & Applications*. ETRA '00. New York: ACM, 2000, pp. 29–36.
11. Riedl MO and Amant St R. Towards automated exploration of interactive systems. *Proceedings of IUI '02*. ACM: New York, 2002, pp.135–142.
12. Salvucci DD and Anderson JR. Automated eye-movement protocol analysis. *Hum Comput Interact* 2001; 16: 39–86.
13. Atkins MS, Moise A and Rohling R. An application of eyegaze tracking for designing radiologists' workstations: Insights for comparative visual search tasks. *ACM Trans Appl Percept* 2006; 3: 136–151.
14. Rayner K. Eye movements in reading and information processing: 20 years of research. *Psychol Bull* 1998; 124: 372–422.
15. Appert C, Beaudouin-Lafon M and Mackay W. Context matters: evaluating interaction techniques with the CIS model. *Proceedings of HCI 2004* (Leeds UK) Springer Verlag, Berlin, Germany, 2004, pp. 279–295.
16. Amant St R and Riedl MO. A perception/action substrate for cognitive modelling in HCI. *Int J Hum Comput Studies* 2001; 55: 15–39.
17. Lohse GL. A cognitive model for understanding graphical perception. *Hum Comp Interact* 1993; 8: 353–388.
18. Cleveland WS. A model for studying display methods of statistical graphics. *J Comput Stat Graph* 1993; 2: 323–364.
19. Casner SM. Task-analytic approach to the automated design of graphic presentations. *ACM Trans Graph* 1991; 10: 111–151.
20. Zhou MX and Feiner S. Visual task characterization for automated visual discourse synthesis. *Proceedings of CHI '98*. ACM: Los Angeles, CA, New York, 1998, pp.392–399.
21. Amar R, Eagan J and Stasko J. Low-level components of analytic activity in information visualization. *Proceedings of IEEE InfoVis '05*. IEEE Computer Society, Washington DC, 2005, pp.111–117.
22. Wehrend S and Lewis CA. Problem-oriented classification of visualization techniques. *Proceedings of IEEE Vis '90*. IEEE Computer Society Press Los Alamitos, CA, 1990, pp.139–143.
23. Card SK, Moran TP and Newell A. The Keystroke-Level Model for user performance time with interactive systems. *CACM* 1980; 23: 396–410.
24. Searle J. *Speech Acts*. Cambridge University Press: Cambridge, UK, 1969.
25. Caelen J. Multimodal human–computer interface. In: Keller E (ed.) *Fundamentals of Speech Synthesis and Speech Recognition*. J Wiley & Sons: Chichester, UK, 1994, pp.339–373.

26. Green TRG. Cognitive dimensions of notation. *People and Computers V*. Cambridge University Press: Cambridge, UK, 1989, pp.443–460.
27. Cockburn A, Gutwin C and Greenberg S. A predictive model of menu performance. *Proceedings of CHI '07*. ACM: San Jose, CA, New York, 2007, pp.627–636.
28. Accot J and Zhai S. Beyond Fitts' law: models for trajectory-based HCI tasks. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '97. ACM: Atlanta, GA, New York, 1997, pp. 295–302.
29. Dragicevic P and Huot S. SpiraClock: A continuous and non-intrusive display for upcoming events. *Extended Abstracts of CHI '02*. ACM: New York, 2002, pp.604–605.
30. MacKay WE. Is paper safer? The role of paper flight strips in air traffic control. *ACM Trans Comput Hum Interact* 1999; 6: 311–340.
31. Grau JY, Nobel J, Guichard L and Gawinoski G. 'Dynastrip': A time-line approach for improving the air traffic picture of ATCOS. *Digital Avionics Systems Conference. DASC '03*. IEEE: Indianapolis, IN, 2003; 22: 5.E.1–511–1.
32. Johansen SA and Hansen JP. Do we need eye trackers to tell where people look? *CHI '06 Extended Abstracts*. ACM: New York, 2006, pp.923–928.
33. Furnas GW. Effective view navigation. *Proceedings of CHI '97*. ACM: New York, 1997, pp.367–374.
34. Johnson J, Roberts TL, Verplank W, Smith DC, Irby CH, Beard M and Mackey K. The Xerox star: A retrospective. *Computer* 1989; 22: 11–26–28–29.
35. Elmqvist N, Dragicevic P and Fekete J-D. Rolling the dice: Multidimensional Visual Exploration using Scatterplot Matrix Navigation Visualization. *IEEE Trans Vis Comput Graph* 2008; 14: 1539–1548.
36. Schlienger C, Conversy S, Chatty S, Anquetil M and Mertz C. Improving users comprehension of changes with animation and sound: an empirical assessment. *Proceedings of Interact '07, LNCS*, Springer Verlag, Rio de Janeiro, Brasil, 2007, pp. 207–220.
37. Heer J and Robertson GG. Animated transitions in statistical data graphics. *IEEE Trans Vis Comput Graph* 2007; 13: 1240–1247.
38. Gibson JJ. *The Senses Considered as Perceptual Systems*. Boston: Houghton Mifflin, 1966.
39. Alva N. *Action in Perception*. The MIT Press: Boston, MA, 2006.
40. Zhang J. The nature of external representations in problem solving. *Cognitive Sci* 1997; 21: 179–217.
41. Zhang J and Norman D. A representational analysis of numeration systems. *Cognition* 1995; 57: 271–295.

# A Visual Perception Account of Programming Languages: Finding the Natural Science in the Art

Stéphane Conversy
Université de Toulouse - ENAC - IRIT
stephane.conversy@enac.fr

## Abstract

*Header:* The textual and visual representation of code should not be considered an art, but should be driven by the capabilities of the human visual system. *Key Insights:* Firm principles which can be relied on to analyze and discuss textual and graphical code representations are still missing. ScanVis, an extension of the Semiotics of Graphics that describes the perception and scanning of abstract graphics, can help describe, compare and invent code representations. This shows that the gap between textual and graphical languages is narrow, and why true visual languages should rely on the capability of the human visual system.

## 1. Introduction

An implicit but important aspect of programming languages is that they must support the production of readable programs [1]: "Programs must be written for people to read, and only incidentally for machines to execute. [2]" Programmers read a program by looking at its 'code', i.e., the representation of the program on the screen, perceptible by the eyes of a human. Both textual and so-called 'visual' representations of programs on the screen employ various graphical 'features': texts, shapes, alignments, colors, arrows, etc. (fig. 1). Those visual features are often considered 'aesthetic sugar' that do not map to semantics (e.g., a colored representation of textual C program), but they can also be part of the syntax (e.g. indented Python code, arrows in state machines, colored Petri-nets).

As with any visual scene, the performance of programmers reading textual or visual programs depends on their performance in perceiving the visual features presented on the screen. However, few works exist that help analyze those features and their impact on performance (an exception is [29]). Instead, programming specialists mention 'aesthetics' or 'personal preferences' [1, 3] and warn about the possible 'danger of religious wars' when dealing with the topic [4]. The use of such terms signals a possible lack of foundation for addressing the phenomenon of code perception and how this may help or hinder programmers' performance.

This paper tackles the principles of programming languages that underpin the practice of code representation: we aim to find the science in the art, rather than finding the art in the science as advocated in [3]. We show how ScanVis, an extension of the Semiotics of Graphics that describes the perception and scanning of abstract graphics, helps describe, compare and generate visual representations of programming languages with respect to human perception. The expected benefits of this work as a scientific point of view are a better understanding of the phenomenon of code perception, the unification of the concepts used in the literature, and accurate definitions of these concepts. The outcome for end programmers would be better designed programming languages and IDEs with respect to this concern.

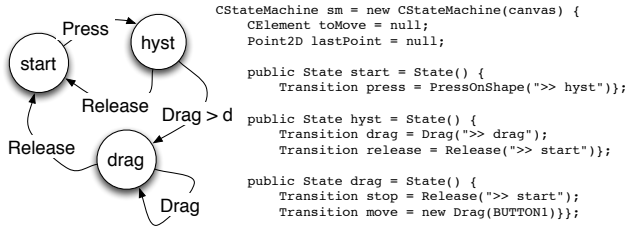



**Figure 1.** Two representations of the same program using various graphical features.

We focus on the representation of 'a single page' of code. Though current trends in this area focus on the management and representation of large-scale programs, representation at the level of the page is overlooked: understanding a single page of code is still required since the very act of programming (i.e., editing code) is often done at this level. In addition, while we appreciate that interaction with code is important [5, 6], we focus solely on the visual perception of code.

## 2. Framework

ScanVis is a model that relies on Semiotics of Graphics. This section presents first the Semiotics of Graphics, then ScanVis.

### 2.1 Semiotics of Graphics

The Semiotics of Graphics is a theory of abstract drawings (i.e., drawings that do not imitate a natural scene) such as maps and bar charts [7]. A part of this theory describes and explains the perceptual phenomena and properties underlying the act of visualizing 2D abstract drawings. The Semiotics of Graphics relies on the characterization of data to be represented (the data type: nominal, ordered, and quantitative), and the perceptual properties of the visual variables used in a drawing, such as color or shape.

Drawings are a set of 2D 'marks' (points, lines or zones) lying over a background. Marks vary according to visual variables such as position (Xpos and Ypos), shape, color, luminosity, size, orientation [7], enclosures and lines that link two marks [8]. Visual variables are characterized by their perceptual properties, and can be: selective – enable a viewer to assimilate and differentiate marks

instantaneously (e.g., all red marks) (fig. 2); ordered – enables a viewer to rank marks perceptually (e.g., from light to dark) (fig. 3); and quantitative – enable a viewer to quantify differences between marks perceptually (e.g. twice as large) (fig. 3).

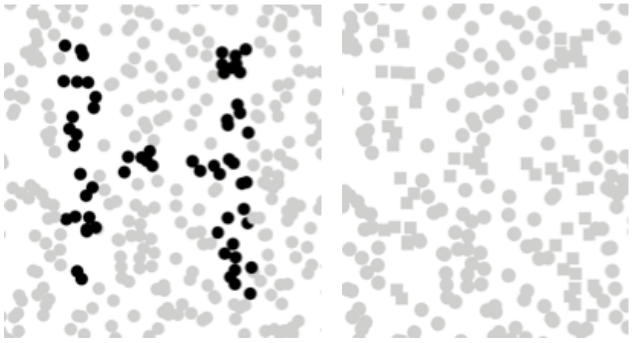


**Figure 2.** By default all marks are circular and light. **Left:** Some marks are dark to produce an H. Luminosity is selective: the H letter emerges because the eye discriminates two groups of marks (light and dark) instantaneously. **Right:** Marks at the same locations, forming the H, are square. Shape is not selective so the H letter does not emerge.

All visual variables but shape and link are selective (fig. 2). All visual variables but shape, link and color are ordered (colors are ordered in a limited spectrum only). Xpos, Ypos, angle, length, size are quantitative to various degrees, as experimentally evidenced [9]. The performance of readers at selecting, ordering or quantifying depends on the number of values differentiable by them (e.g., 5 levels of luminosity for selection, 20 levels of luminosity for order), the difference between each value (the less the worse) and the spatial distance between marks (the more the worse).

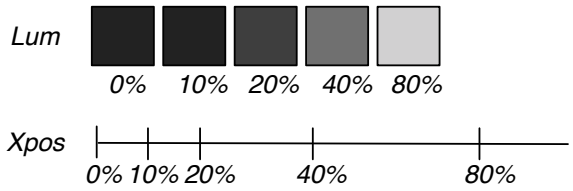



**Figure 3. Top:** Luminosity is ordered but cannot be perceived quantitatively. **Bottom:** Position is quantitative: one can perceive the ratio and the difference between various X positions (pos. 80% is 2x pos. 40% and 4x pos. 20%).

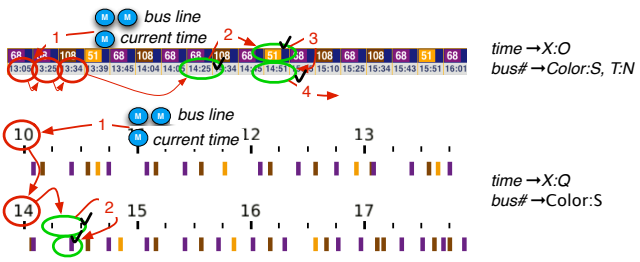



**Figure 4.** Two visualization and the visual operations for the task "find how long I have to wait for the next bus." [6] Right: mapping between data and visual variables (e.g. '$time \rightarrow X : O$' means that 'time' is mapped to Xpos in an Ordered manner).

### 2.2 ScanVis

The Semiotics of Graphics may help design a representation that enables users to perceive multiple information elements at a single glance. Nevertheless, however well designed a representation is, it cannot be absolutely efficient: a representation may be well suited to a particular task, but may not be suitable for all tasks a user's activity requires. For such tasks, instead of perceiving the representation at one glance, the user falls back to scanning the representation to discover information. For example, fig. 4 shows two different representations of a bus schedule. The overlaid arrows and circles depict the visual scanning required on each to answer the same question: "how long will I wait for the next bus?". Depending on the representation (in this case ordered or quantitative), the amount and the nature of visual scanning operations will differ.

ScanVis is a descriptive model of this kind of representation scanning [6]. It enables a design to analyze and assess a representation effectiveness with respect to a task. ScanVis relies on the decomposition of representation scanning into elementary visual operations: *enter into the representation* by transforming the conceptual task at hand ("how long will I wait?") into a reading task ("find the time corresponding to my bus lines"), *seek a subset of marks* ("find the marks corresponding to my bus lines?"), *seek and navigate among a subset of marks* ("navigate into a row of text representing time in a time-table"), *unpack a mark and verify a predicate* ("what datum does this position reflect, and does it answer my question?"), *exit from the representation* ("this bus passes at this time, I need to compute the waiting time"), and *memorize* information ("I have to wait 2 minutes for this bus; remember this to compare with another bus"). Given a task and a representation, a designer can infer the required sequence of visual operations to accomplish it.

ScanVis' elementary operations may be facilitated by the use of adequate visual properties as described by Semiotics of Graphics e.g. *selective* visual variables to support *seeking and navigating among a subset marks*: if one wants to find out the next bus #51 in the representation at the bottom of fig. 4, one can visually *select* a subset of marks that are yellow (a selective variable) and that lie in the area whose position (a selective variable) corresponds roughly to the current time. One can then scan through the marks of this subset and hop from mark to mark until the next bus is found.

## 3. Application to Programming Languages

ScanVis and Semiotics of Graphics (together referred as 'the framework') have already been applied to charts or visualizations of information. The remainder of this paper is devoted to their application to programming languages. In order to help the reader assess the significance of the proposed framework, we will describe its descriptive power (what are the phenomena that the framework captures?), its comparative power (how can it help assess or compare particular code representations?) and its generative power (how can it help explore the design space of code representation?).

As illustrated in the ScanVis section, assessing a particular visual representation of a program requires identifying the set of reading tasks performed by the programmer. In the following, we present a number of visual representations together with tasks. Since reading tasks have seldom been clearly stated in the literature, we have had to devise them with reference to interesting aspects of the representations. We believe that the tasks are appropriate but we do not claim perfect validity; the reader of this article may disagree. In fact, a desired outcome of this paper is to initiate work on the elicitation of reading tasks. The tasks presented here are a first step in that direction.

## 4. Describing Visual Features Of Languages

The goal of this section is to show how popular sayings about code (e.g. "as an art") can be appropriately deconstructed and translated to the concepts and vocabulary (in italics) of the framework. If correct and comprehensive, such a translation is the first step toward the validation of the significance of the framework.

### 4.1 Visually structuring the code

*"Lots of Irritating Superfluous Parenthesis."* Lisp uses parentheses to structure code. Lists are designated with spaced expressions surrounded by opening and closing parentheses. Function composition uses compound parenthesizing.

```
(defun fac (n) (if (<= n 1) 1 (* n (fac (- n 1)))))
(defun fac ◁n▷ ◁if ♢<= n 1♢ 1 ♢* n ♡fac ☞- n 1☜♡♢▷)
```

**Figure 5.** Delimiters varying in shape. Top: parenthesis, bottom: other shapes.

Lisp is reputedly difficult to read ([10] p65). The difficulty comes partly from the fact that list boundaries are depicted with two *shapes* (opening and closing parentheses, see fig. 5, top) which prevents fulfilling the task "figure out the [lisp] expressions" efficiently. This can be explained with the *selectivity* concept: since shape is *non-selective*, use of parentheses prevents the perception of Lisp expression boundaries at a single glance and forces the programmer to *scan* the code linearly to discover them. The bottom line of fig. 5 uses a unique boundary shape according to the level of depth of enclosure. One may think that such a representation could help the reader match the boundaries of expression since the use of unique symbols should prevent the reader to mismatch opening and closing boundaries. However, it is not better than the representation with parenthesis because of the same phenomenon: the *non-selectiveness* of symbols prevents matching at a single glance e.g. finding the diamond which closes the multiply expression is difficult and requires a careful horizontal scan.

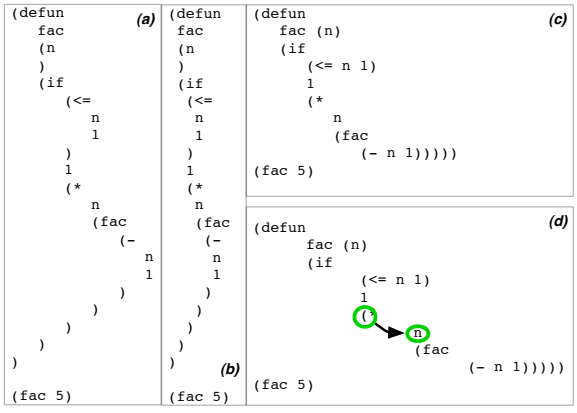



**Figure 6.** Delimiters varying in Xpos and Ypos, which are both selective visual variables.(b) A smaller difference between Xpos values hinders selection. (c) Improving Xpos selectiveness by shortening spatial distances in Ypos or (d) with a larger indentation, at the expense of visual scanning, depicted with circles and arrows.

*"Indentation makes structure obvious."* In fig. 6 (a) the level of nesting depth is mapped to the *Xpos visual variable*. Matching parentheses are "vertically aligned", which is another way of expressing an *assimilation of Xpos values*. Since Xpos is *selective*, the perception of expression boundaries is better than when using a shape. Selectivity depends on the *amount of difference* between values: shrinking the size of the indentation lowers the selective ability of Xpos (b). Reserving a line for a closing parenthesis alone lengthens the Ypos *spatial distance* with the matching opening parenthesis and weakens the selective property of the Xpos visual variable (i.e., it is difficult to perceive vertical alignment) (a and b); ignoring the parenthesis matching problem altogether shortens the distances and improves Xpos selectivity (c); more indentation improves selective perception of Xpos (d). However, the assumed improvement is supposedly accomplished at the expense of longer *scanning* from the beginning of a block to its first instruction, as experimentally assessed in [20]. Note that since Xpos is ordered, such a representation also facilitates the task "figure out the hierarchy of expressions".

*"LabView's G language is intuitive."* G mixes large boxes that *enclose* other objects to specify a hierarchical structure (fig. 7), and *links* that connect the components inside and outside boxes (see [24] for more details). Enclosures may be "intuitive", but a more appropriate qualification is that they are *selective*: one can grasp in a single glance which elements are part of a parent. Enclosure is also *ordered* and help perception of a containment hierarchy.

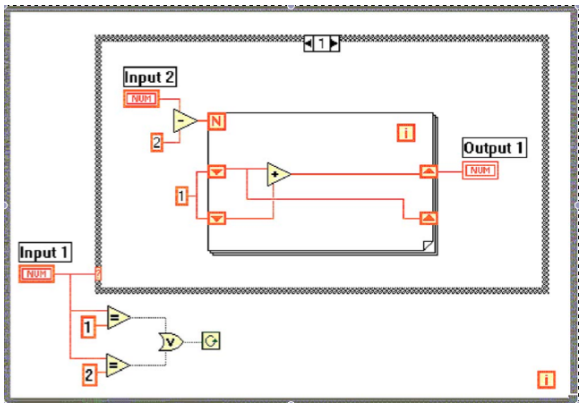



**Figure 7.** G language from LabView.

*"Syntax highlighting improves readability."* Fig. 8 shows a 'syntax-colored' textual representation of Java code in the NetBeans editor. Blue glyphs correspond to reserved keywords of the Java language and gray ones to comments. A yellow background corresponds to a variable on which the mouse pointer points.

```
// replicable pseudo random generator
Random rpos = new Random(456);
Random r = new Random(321);
double[] sizes = new double[6];
double a = 10, b = 5;
for (int i = 0; i < sizes.length; ++i) {
    sizes[i] = a * i + b;
}
float[] tricol = new float[3], rgb, lch;
lch = tricol;
lch[0] = 40;
lch[1] = 100;
lch[2] = 45;
Color c1 = srgb.fromLCHtoColor(lch);
[…]
System.out.println("[debug] color is"+c1);
// compute each symbol hue
for (int i=0; i<hue_symbol.length(); ++i) {
   lch[2] = (float)(i*360./hue_symbol.length());
   colors[i] = srgb.fromLCHtoColor(lch);
   hue_shapes.add(buildShape(g, hue_symbol.substring(i,i+1), w, h));
}
```

**Figure 8.** Colored editor.

Coloring all appearances of a variable the mouse is pointing at does make sense from a programming task point of view: this enables the programmer to efficiently identify all occurrences of this variable thanks to *selectivity*. Similarly, adding a colored background to a brace enables the programmer to quickly see where the corresponding brace is and assess the scope of a block. The gray color is lighter than the other ones: since luminosity is *selective*,

this enables user to rapidly *assimilate* and *differentiate* code from comments, and rapidly *navigate* between sections of the code. This removes the need to *scan* the beginning of a line to check whether it begins with two slashes, a much more demanding visual task since shape (‘//’) is not *selective*. In addition, the *order* of luminosity indicates an order of importance between code, comments, and background.

### 4.2 Understanding control flow

*“The control flow in C is visible.”* In a block of C instructions, a sequence of texts separated by semicolons denotes a sequence of instructions (9). However, as a shape, semicolons are not selective, and do not help discriminate between instructions. Often, instructions are organized one per line. In this case, the Ypos visual variable maps to the ordered sequence of the program counter. This helps the programmer visualize the evolution of the program counter path by *scanning* the textual instructions vertically. Thus, the task “given an particular instruction, what is the next instruction to be executed?” is efficiently supported by the representation since it uses a *selective*, *ordered* variable. However, function calls and gotos are not as visible since they are indicated by name (a *shape*), which is not a selective visual variable. In order to perform the “next instruction?” task for a call instruction, the reader has to *scan* the representation and *seek* the answer.

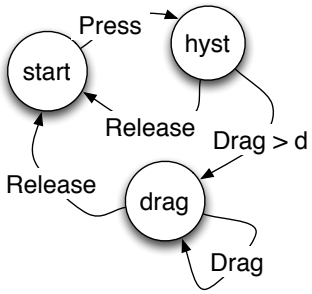
```
extern void printf(char*,...);

int fact(int n);

int main(int argc, char* argv[]) {
  int res = fact(argv[1]);
  printf("fact %s: %s\n",argv[1],res);
  return 0;
}

int fact(int n) {
  int res=1;
  while (n) {
    res *= n;
    n-=1;
  }
  return res;
}
```

*function decl → Y:O*
*function def → Y:N*
*instruction flow → Y:O*
*loop/branch/jump → Text:N*

*call function → Text:N*
*block → X:O*
*block → shape(X,Y)'{''}':N*

**Figure 9.** C language classification.

*“Arrows make the instruction flow explicit”* Fig. 10 shows a circle-and-arrow description of a Drag’n’Drop interaction with hysteresis [11]. There are three states (‘start’, ‘hyst’ and ‘drag’), one transition from state ‘start’, and two each from states ‘hyst’ and ‘drag’. The instruction flow is depicted with marks: arrows. To fulfill the task “figure out the flow”, a reader must *seek a subset of marks* (*links*) and *navigate* visually by following arrows, which may be slow especially when arrows are numerous and entangled.




*state → circle:S*
*transition → link:S*
*out transition → link/L:S*
*in transition → link/L:S*
*instruction flow → link:scan*

*state name → text:N*
*transition clause → text:N*
*PC → N/A*

**Figure 10.** Box-and-arrow representation of a state machine.

Code Bubble is an IDE that presents code with function snippets inside individual windows resembling ‘bubbles’ [12]. Users can juxtapose bubbles that contain related code. One use is to display the code of a callee in a bubble to the right of a bubble containing the caller. Hovering over a bubble highlights the connections and code lines that lead to it by changing the color or luminosity of the links. This turns a *non-selective* variable (link) *selective* (colored link) and helps readers figure out the flow and navigate between instructions that belong to related functions.

*function decl → Y:O*
*function def → Y:N*
*instruction flow → Y:O*
*loop/branch/jump → Text:N*

***call function → Arrow:N***
*block → X:O*
*block → shape(X,Y)'{''}':N*

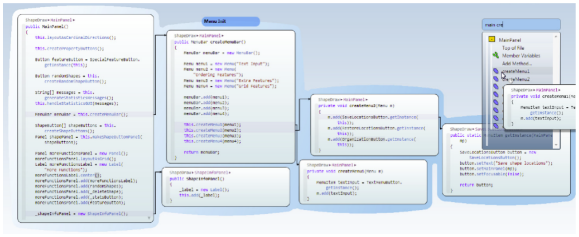


**Figure 11.** Code Bubble classification.

```
0&>:1-:v v *_$.@
  ^     _$>\:^
```

*instruction flow → X,Y:O+Symbol:N*
*loop/branch/jump → Symbol:N*
*call function → N/A*
*block → Symbols Containment:N*

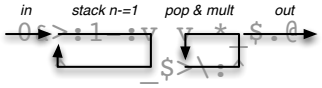



**Figure 12.** Factorial in befunge (top); explanation of the flow (bottom).

*“Befunge is an esoteric language”* Befunge is a 2D textual language in which the flow is indicated by the four *shapes* <, >, ^, and v, which resemble arrows pointing in the four cardinal directions. Branching is specified by - (equivalent to < if the condition is true and to > otherwise) and | (equivalent to ^ if the condition is true and to v otherwise). However, not only is the flow not graspable at once (real arrows and links help a little in the bottom part of the picture), but directional shapes are not *selective* and cannot be perceived instantaneously. As such, Befunge is perhaps not so esoteric since it can be considered a missing link between textual and visual languages. This illustrates the unifying aspect of the framework.

### 4.3 Understanding functionality: “Icons are easier to use than text”

Icons are often considered easier to interpret than text. Fig. 8 in [26] illustrates the use of an ‘analog’ [26] iconic vocabulary in LabView’s control-flow structures. In this case, icons are differentiated with *shapes* (arrowheads, page corners, spirals, ‘N’ and ‘I’). When icons vary according to shape only (a non-selective variable), one can only perform a slow *elementary reading* of a scene. In other visual languages, icons vary in shape but also along other visual variables, which may turn them selective. For example, the third line of fig. 13 uses a set of shapes with which *selection* and *ordering* seem to ‘work’: this is because they do not contain the same number of pixels and exhibit different levels of *luminosity*, a selective variable.

## 5. Comparing Code Representations

This section shows how the ScanVis plus Semiotics of Graphics framework helps compare code representations with respect to tasks.

### 5.1 Luminosity, color and position of enclosing symbols

In fig. 13, the first line maps depth of nesting to unique colors. Since color is selective, this enables the reader to *assimilate* at one glance all parentheses with the same level of depth. However, one has to wonder if the task "assimilate level of depth" is worth facilitating: even if a reader correctly detects each opening and closing parenthesis, one must remember the discovered structure to make sense of it. If an appropriate visual variable was used instead, the programmer could use that as an externalization of memory to recall the structure by accessing it immediately, in one glance. For example, the second line uses luminosity alone. Luminosity is *selective* (and helps match parentheses), and *ordered* (helps perceive relative depth). One cannot perceive the exact depth since as opposed to Xpos luminosity is not *quantitative*. However, ordering may be sufficient for the task at hand.

```
(defun fac (n) (if (<= n 1) 1 (* n (fac (- n 1)))))
(defun fac (n) (if (<= n 1) 1 (* n (fac (- n 1)))))
(defun fac ◁n▷ ◁if ✳<= n 1✳ 1 ✳* n ✳fac ✷- n 1✷✳✳▷)
```

**Figure 13.** Delimiters varying in hue, luminosity, and shape+lum.

### 5.2 Ypos versus Arrows

As we have seen above, instruction flow can be depicted using Ypos or links and arrows. Links and arrows are not selective visual variables: the reader is forced to follow the chain of links to figure out the flow (fig. 14-a). This can be supplemented with alignment cues, e.g., using the selective property of Ypos. In this case, the visualization is equivalent to indented code in a C program (fig. 14-b). It is not necessary to show the arrows between successive instructions as this is redundant with the Ypos-aligned representation (fig. 14-c) However, keeping an arrow for the loop helps the reader *scan* up to the beginning of a loop, similar to box-and-arrow languages. Scratch [13] is a visual language with connectors on blocks suggesting how they should be put together. The connectors are similar to the arrows: they guide a reader following the instruction sequence (fig. 14-d). The start of the loop can be perceived selectively with color and containment. These examples show how different representations can be unified with the same underlying principles.

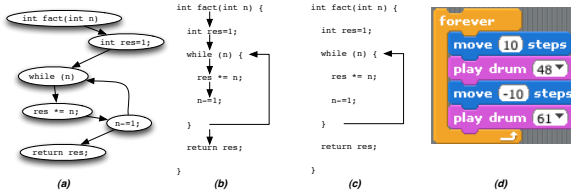



**Figure 14.** Arrows could have been used in C (b), as in box-and-arrow languages (a). Since arrows are redundant with the ordered Ypos visual variable, they can be removed, except for the loop (c). Scratch uses similar visual variables (d).

Arrows are often said to be an explicit representation of instruction sequence. To be more precise, they are an explicit representation of sequence direction. However, they are no more explicit on the order of the sequence than Ypos, since the selective visual variable Ypos explicitly shows sequence already (both in the C version and the Scratch version).

### 5.3 Comparing representations with multiple, more demanding tasks

Code representations are used to fulfill multiple reading tasks. When comparing them, it is helpful to enumerate a realistic set of tasks and assess how each representation rates with respect to each task.

```
CStateMachine sm = new CStateMachine(canvas) {
    CElement toMove = null;
    Point2D lastPoint = null;

    public State start = State() {
        Transition press = PressOnShape(">> hyst")};

    public State hyst = State() {
        Transition drag = Drag(">> drag");
        Transition release = Release(">> start")};

    public State drag = State() {
        Transition stop = Release(">> start");
        Transition move = new Drag(BUTTON1)}};
```

*state → X:S?*
*transition → X:S?*
*out transition → text:N*
*in transition → text:N*
*next state → text:N*

*state name → text:N*
*transition clause → X:S*
*instruction flow → Y:O*

**Figure 15.** 1D Text representation of the state machine.

Fig. 15 shows the SwingState code describing the same Drag'n'drop interaction as in Fig. 10 [11]. SwingStates is a textual language for describing state machines [14]. It relies on Java's anonymous class facility to be embedded seamlessly in regular Java code. The code is indented to facilitate perception of the states, the transitions from each state, and the clauses associated with the transitions.

Fig. 16 compares the visual operations required for the task "what are the 'out' transitions for a particular state?" in the circles-and-arrows and SwingStates representations. In both, readers need to seek and navigate among states until they find the state of interest, then seek all transitions leaving this state. With circles-and-arrows, one can consider that large white circles are selective compared to other marks because of their size and luminosity. In the SwingStates code, indentation is also selective. Hence both representations help seek a subset of marks. Finding 'out' transition is more efficient in SwingStates code since all transitions are out transitions. With circle-and-arrows, one has to differentiate between links with and without arrowheads, laid around the circles. Links without arrowheads may be more difficult to differentiate than other marks.

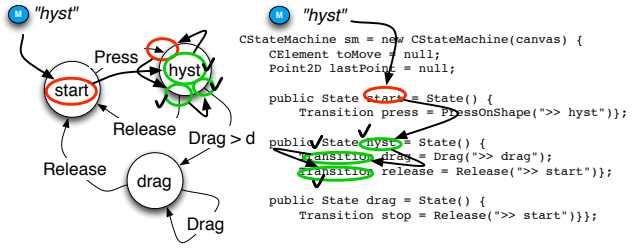



**Figure 16.** ScanVis for the task: "what are the 'out' transitions for state 'hyst'?" .

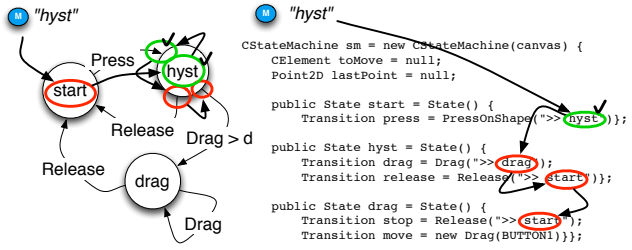



**Figure 17.** ScanVis for the task: "what are the 'in' transitions of state 'hyst'?"

Fig. 17 illustrates the visual operations required for the task "what are the 'in' transitions for a particular state?" For the circle-and-arrows representation, the operations are almost identical to the

operations required in the previous task. Finding the 'in' transition may be facilitated by the fact that arrowheads are dark and thus selective. With the SwingState code, the visual operations are very different: one has to find the name of the target states inside the transitions. Most of those names are on the right edge of the code, which helps seek and find them. Still, since they are textual, it may be difficult to navigate without risk of missing a name.

## 6. Generating New Code Representations

This section introduces a set of design principles that can be used to make new code representations emerge and illustrates them with a number of examples. We devised the design principles by examining how existing representations improve over previous ones.

*Identify the task and apply selectivity only where needed.* In the colored code fig. 8, using color for all keywords may not be related to any task the programmer needs to accomplish (e.g., find all 'for' loops). Of course, one can argue that the distinction helps assess that a keyword has been recognized as the programmer types it and that no lexical error has been made. However, fulfilling this task does not require a selective variable such as color. Instead, elementary reading with a non-selective variable such as a shape, or a typeface (e.g., 'unrecognized' in 'courier') is sufficient. This would reserve color, a scarce resource, for a more important use.

*Try swapping visual variables.* One way to generate representations is to explore the design space of code representation by swapping visual variables for unused ones. Fig. 18 illustrates alternative representations using size and Ypos as visual variables instead of color. Since these visual variables are selective and ordered, they help the reader visualize the structure of the code, similarly to the more traditional use of the Xpos visual variable (indentation).

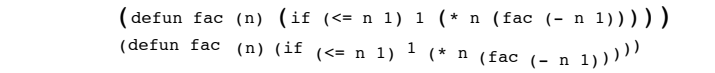



**Figure 18.** Using size and Ypos as visual variables.

*Shorten spatial distance.* As mentioned previously, reducing spatial distance may improve selectivity. Fig. 6(c) shows a representation that shortens spatial distance, but does not support parenthesis matching, which can be annoying when trying to add a missing parenthesis. Fig. 19 is a representation that shortens the spatial distance while enabling easy parenthesis matching. Remarkably, while the parentheses match perceptually according to the Xpos visual variable, they do not match conceptually: for example, the opening parenthesis at the beginning of the 'defun' function conceptually matches the rightmost closing parenthesis on the penultimate line of the code, but is aligned with the closing parenthesis of the call to factorial. This illustrates that perception can prevail over the conceptual model, as long as semantics is preserved.

```
(defun
   factorial (n)
   (if
      (<= n 1)
      1
      (*
         n
         (factorial
            (- n 1)
)  )  )  )

(factorial 5)
```

**Figure 19.** Shortening spatial distance: parentheses match perceptually, but do not match conceptually.

Another representation that reduces distance is shown in Fig. 20. With a debugger, the user can step inside the call of a function. This can be performed with a toggle arrow: when toggled, the code of the function unfolds under the call of the function. A similar feature could be used for static code; this would help understand how functions compose without the need to memorize the code surrounding the call of a function and to switch visually between the distant representations of the two functions.

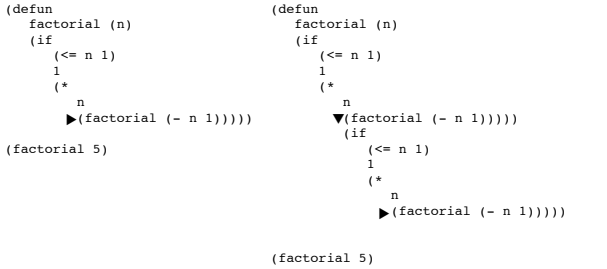



**Figure 20.** 'Debugger view' of code.

One asset of such a 'tree-view' is that it shortens the distance between instructions before the call and instructions at the beginning of the function being called. However, it also expands the distance between the instructions before the call and after the call, especially when multiple functions are deployed. A 'browser' view ala SmallTalk can help show details and contexts of the call and shorten both distances (Fig. 21). CodeBubbles can be seen as an attempt to shorten the distance between calling and called code.

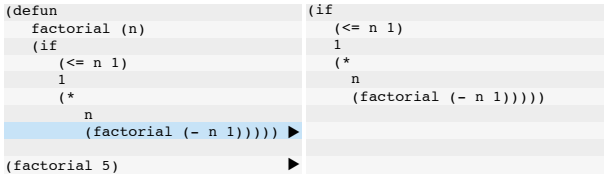



**Figure 21.** Browser view of the 'factorial' function.

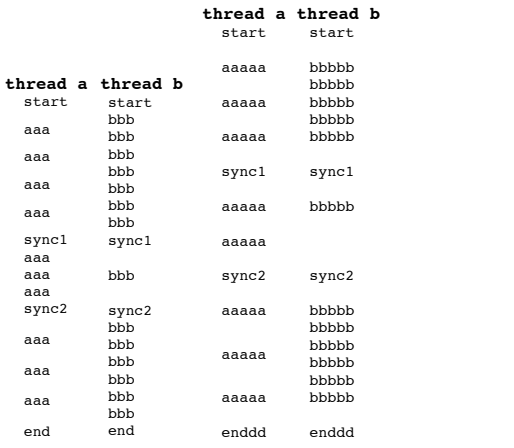



**Figure 22.** Left: Ypos used as a selective variable: instructions are aligned when synchronized, and misalign when not synchronized. Right: Ypos used as a quantitative variable: the number of cycles is mapped to the distance between instructions (aaaaa: 2 cycles, bbbbb: 1 cycle).

*Explore and leverage properties of visual variables.* In a typical imperative language, Ypos is used in an ordered but not quantitative manner. Since the distance between instructions has no meaning, a representation could vary distances to misalign statements and align synchronization statement only. In Fig. 22-left, the selectivity of the Ypos variable helps show at a glance the synchronization points and removes false information conveyed by perfectly aligned statements. Distance can also be used to convey quantity. Fig. 22-right illustrates a representation that uses Ypos as a quantitative variable to depict two concurrent sequences of instructions. The number of cycles taken by each instruction is mapped to the Ypos dimension. The larger the space after an instruction, the larger the number of cycles it takes to execute it. This gives a sense of the time spent on some parts of code, and can help balance the instructions in order to minimize wasted cycles while waiting for the concurrent process when synchronization is needed.

## 7. Threats To Validity

The proposed framework relies on models, and as such is a simplification of reality. Even if the framework allows us to describe a number of the perceptual phenomena underlying the perception of code, some important phenomena may not have been identified because of the limited capability of the framework, or because their explanation or cause is different (a hammer and nail problem). Visual perception is complex and some visual operations may be bypassed because of specific conditions such as layout or the number of items involved. Further, code representation is not the only factor that contributes to program understanding. Other cognitive factors, such as learning, expertise, API usability and documentation [19] contribute to program understanding, and may influence the way the user perceives or scans the code.

## 8. Related Work

Reading code is a complex process that involves many aspects. We have selected a number of works that address formatting, performance at reading, differences between textual and visual languages, and frameworks to analyze them.

### 8.1 Formatting and pretty-printing

'Formatting well' is often advised and discussed in early fundamental papers about programming languages (e.g., the discussion in [15]): "Code formatting is about communication, and communication is the professional developer's first order of business" [4]. In a recent work, formatting is still referred to as an 'art' [3]. Actually, the problem of program representation goes well beyond code formatting and refers to the more general problem of the visual perception of the code by the programmer.

### 8.2 Performance at reading programming languages

A number of visual designs have been proposed to improve reading performance [16–19]. Indentation length has been experimentally shown to have an impact on the comprehension of code: 2- and 4-space indentation makes readers better at understanding the code than 6-space indentation, for both novice and expert readers [20]. Eye tracking has been used to observe programmers but only to measure switching between a view of the code and a view presenting an animated algorithm [21].

Moher et al. observed that "performance was strongly dependent to the layout of the Petri nets. In general, the results indicate that the efficiency of a graphical program representation is not only task-specific, but also highly sensitive to seemingly ancillary issues such as layout and the degree of factoring" [22]. Green et al. found that textual representations outperformed LabView for each and every subject [23]. Their explanation is that "the structure of the graphics in the visual programs is, 'paradoxically', harder to scan than in the text version". LabView and its G language have been studied "in the wild" [25]. Respondents declared that G is easier to read than textual programming languages since it provides an overview (a gestalt view) and clarifies structure. However, respondents also say that it is very easy to create messy, cluttered, hard to read spaghetti code and that sequence structures tend to be cryptic or obscure.

### 8.3 Differences between textual and visual languages

Researchers have already wondered where the actual differences between textual and visual languages lie. In [26] Petre argues that the differences in effectiveness between textual and visual languages "lie not so much in the textual-visual distinction as in the degree to which specific representations support the conventions experts expect." As Petre observed, programmers can find gestalt patterns in textual representations [26]. Much of "what contributes to comprehensibility of a graphical representation is not part of the formal notation but a 'secondary notation': layout, typographical cues and graphical enhancements." Petre adds that "the secondary notations (e.g., layout) are subject to individual skills (i.e., learned ones) and make the difference between novices and experts. What is required in addition is good use of secondary notation, which like 'good design' is subject to personal style and individual skill" [26]. We take an alternative point of view: we argue here that even if skills can be learned, the basic visual capability of humans is enough to explain much of the ease or difficulty programmers experience in deciphering a program.

### 8.4 Analysis frameworks

There have been several attempts at building metrics for software readability. In the metric from [27], a few features can be considered perceptual (commas, spaces, indentation), but most are based on the semantics of the code. The cognitive dimensions of notation (CDN) is a framework that helps designers analyze interactive tools, including programming environments and languages [5]. CDN targets cognitive and interactive aspects as opposed to perceptual aspects: the graphic and perceptual concerns are addressed partly in the secondary notation and visibility dimensions. Gestalt is a well-known framework that explains the phenomena underlying pattern perception. Gestalt can be used to explain how programmers may perceive patterns in their code, but we found that Gestalt could not report about all perception phenomena. So-called pre-attentive features also have a role in the perception of code [28]. The Semiotics of Graphics encompasses pre-attentive features and addresses other levels of perception than Gestalt.

The Physics of Notations framework focuses on the perceptual properties of notations [29] and partly relies on the Semiotics of Graphics. Our work offers a finer and more complete account of how the Semiotics of Graphics apply to graphical *and* textual code representation. Nevertheless, the Semiotics of Graphics alone cannot be used to assess code representation: the use of ScanVis and its emphasis on image scanning and precise task elicitation (e.g. "match parenthesis" vs "figure out the hierarchy of expressions", or "what is the next bus?" vs "how long will I wait?") makes the analysis finer and helps design more efficient representations.

## 9. Why does it matter?

We have successfully captured a large set of phenomena pertaining to code representation at the level of "the page of code" with the Semiotics of Graphics and ScanVis. For a framework to describe and corroborate existing phenomena is a first level of validation. Furthermore, this success shows that there are important issues at stake when a programmer reads code. It shows that code

representation is not about aesthetics but performance, and should not be an art but a science following principles from visual perception. To foster understanding of a program, a representation of code that follows those principles is not accessory, but mandatory. Therefore the account presented in this paper extends the set of important aspects underlying programming languages: lexical (what concepts are), syntactical (how concepts articulate), semantic (what concepts mean), but also perceptual (how efficiently concepts are represented, with respect to programming tasks). This should be a concern for all educated computer scientists and programmers, be they academic or practitioner, as much as basic knowledge about programming such as "functional and imperative programming", or "static and dynamic typing".

Another perspective opened by this work is the unification of existing concepts. Unifying concepts has been a traditional goal in science (e.g., Maxwell's equations unifying electricity and magnetism, or the Curry-Howard correspondance between types and proofs) because this may lead to important discoveries and insights. Here the framework brings together many aspects of visual layout and appearance of programming languages and contradicts the traditional opposition between visual and textual languages. It also contradicts the usual wisdom that visual languages are by essence better than textual languages: most textual languages are displayed using positional variables and thus may use the perceptual system efficiently, while some so-called visual languages may use visual variables (icons (shapes), links) quite inefficiently. This should be of interest for any educated computer scientist, including software engineers who often use various UML diagrams (a visual language) to document their software.

In addition, the fact that the framework is comparative should encourage programmers to expect justifications from language designers. They should be compelled to explain why and how the language designed is better than another with respect to objective criteria. This should diminish the risk of "religious wars", since using a shared, consistent set of reference concepts would make the comparisons and justifications claims better supported.

Finally, the generative power of the framework may enable language designers to find new ways of representing the code. The examples and the provided design principles can help explore the design space of code representation. Even if a complete and detailed method is still missing, what the framework suggests is that such a method should use a "Programmer-Centered Design" approach: it should emphasize the act of designing with end-programmers' tasks in mind, and not designing the representation alone. A follow-up on this work is thus both to gather the reading tasks that are supposedly supported by today's language representation and the overlooked reading tasks that a programmer constantly fulfills in order to program. The outcome for end programmers would be more efficient code representation.

## References


[1] Raymond, D. 1991. Reading source code. In Proc. of the 1991 conference of the Centre for Advanced Studies on Collaborative research (CASCON '91), Ann Gawam, Jan K. Pachl, Jacob Slonim, and Anne Stilman (Eds.). IBM Press 3-16.

[2] Abelson, H. and Sussman, G. 1996. Structure and Interpretation of Computer Programs (2nd ed.). MIT Press.

[3] Green, R. and Ledgard, H. 2011. Coding guidelines: finding the art in the science. Commun. ACM 54, 12 (December 2011), 57-63.

[4] McConnell, S. 2004. Code Complete, 2nd edition. Microsoft Press.

[5] Alan F. Blackwell and Thomas R.G. Green. (2003) Notational Systems - the Cognitive Dimensions of Notations framework. In HCI Models, Theories, and Frameworks, Morgan Kaufmann, pp. 103-134.

[6] Conversy, S., Chatty, S., Hurter, C. Visual Scanning as a Reference Framework for Interactive Representation Design. In Information Visualization, 10, pages 196-211. Sage, 2011.

[7] Bertin, J. (1967) Sémiologie Graphique - Les diagrammes - les réseaux - les cartes. Gauthier-Villars et Mouton & Cie, Paris.

[8] Card, S.K., Mackinlay, J.D., Shneiderman, B., Readings in Information Visualization: Using Vision to Think. San Francisco, California: Morgan-Kaufmann, (1999).

[9] Cleveland, W., McGill, R., Graphical Perception and Graphical Methods for Analyzing Scientific Data. Science, New Series, Vol. 229, No. 4716 (Aug. 30, 1985), pp. 828-833.

[10] Steele, G. and Gabriel, R. 1996. The evolution of Lisp. In History of programming languages—II, Thomas J. Bergin, Jr. and Richard G. Gibson, Jr. (Eds.). ACM, New York, NY, USA 233-330.

[11] Conversy, S. Improving Usability of Interactive Graphics Specification and Implementation with Picking Views and Inverse Transformations. In Proc. of VL/HCC, pages 153-160. IEEE, 2011.

[12] Bragdon, A., Zeleznik, R., Reiss, S., Karumuri, S., Cheung, W., Kaplan, J., Coleman, C., Adeputra, F. and LaViola, J. 2010. Code bubbles: a working set-based interface for code understanding and maintenance. In Proc. of CHI '10, ACM, 2503-2512.

[13] Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y. 2009. Scratch: programming for all. CACM, 52, 11, 60-67.

[14] C. Appert and M. Beaudouin-Lafon. (2008). SwingStates: Adding state machines to Java and the Swing toolkit. Journal Software Practice and Experience. 38, 11 (Sep. 2008), 1149-1182.

[15] P. J. Landin. 1966. The next 700 programming languages. Commun. ACM 9, 3 (March 1966), 157-166.

[16] Clifton, M. 1978. A technique for making structured programs more readable. SIGPLAN Not. 13, 4 (April 1978), 58-63.

[17] Crider, J. 1978. Structured formatting of Pascal programs. SIGPLAN Not. 13, 11 (November 1978), 15-22.

[18] Ramsdell, J. 1979. Prettyprinting structured programs with connector lines. SIGPLAN Not. 14, 9 (September 1979), 74-75

[19] T. Tenny. 1988. Program Readability: Procedures Versus Comments. IEEE Trans. Softw. Eng. 14, 9 (September 1988), 1271-1279.

[20] Miara, R., Musselman, Navarro, J. and Shneiderman, B. 1983. Program indentation and comprehensibility. CACM 26(11), 861-867.

[21] Bednarik, R. and Tukiainen, M. 2006. An eye-tracking methodology for characterizing program comprehension processes. In Proc. of the 2006 symp. on Eye tracking research & applications (ETRA '06). ACM, New York, NY, USA, 125-132.

[22] Moher, T.G., Mak, D.C., Blumenthal, B., and Leventhal, L.M. Comparing the comprehensibility of textual and graphical programs: The case of Petri nets. In Empirical Studies of Programmers: 5th Workshop. Ablex, 1993, 137-161.

[23] T. R. G. Green & M. Petre (1992) When visual programs are harder to read than textual programs. Proc. of the 6th European Conference on Cognitive Ergonomics (ECCE 6), pp. 167-180.

[24] Whitley, K., Novick, L. and Fisher, D. 2006. Evidence in favor of visual representation for the dataflow paradigm: An experiment testing LabVIEW's comprehensibility. Int. J. Hum.-Cmp. Stud. 64(4), 281-303.

[25] Whitley, K. and Blackwell, A. Visual Programming in the Wild: A Survey of LabVIEW Programmers', Journal of Visual Languages & Computing, 12(4), Aug. 2001, p435-472.

[26] Petre, M. 1995. Why looking isn't always seeing: readership skills and graphical programming. Commun. ACM 38, 6 (June 1995), 33-44.

[27] Buse, R. and Weimer, W. 2008. A metric for software readability. In Proc. of ISSTA '08. ACM, 121-130.

[28] Treisman, A. (1982). Perceptual grouping and attention in visual search for features and for objects. Journal of Experimental Psychology Human Perception and Performance, 8(2), 194-214.

[29] Moody, D. 2009. The 'Physics' of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Trans. Softw. Eng. 35, 6 (November 2009), 756-779.