AN APPROACH FOR GENERALIZING FOCUS+CONTEXT TECHNIQUES

Francisco J. Almeida-Martínez
Universidad Rey Juan Carlos
francisco.almeida@urjc.es

Jaime Urquiza-Fuentes
Universidad Rey Juan Carlos
jaime.urquiza@urjc.es

ABSTRACT

Many efforts have been dedicated to solve the problem of handling large scale workspaces. Numerous techniques have been designed, most of them have lots of specific implementations. Also, there exist some generic implementations that support the development of interfaces using these techniques. The focus+context family lacks of this kind of generic solutions. In this paper we describe a focus+context technique called R-Zoom and its generalization. From this experience we propose a generalization framework that could help to design generalizations for other focus+context techniques. This framework is based on the independence from the individual nature of the elements and the parameterization of the layout and manipulation features.

KEYWORDS

Information visualization, Generalization, Large scale workspaces.

1. INTRODUCTION

The information visualization field has contributed with many solutions to the problem of handling large scale workspaces. Many techniques have proved to be useful. Some of them even in different domains than the original, e.g. TreeMaps [1] and its different commercial uses. Once that an interface developer has decided to use one of these techniques in a domain different from the original, he has two complex possibilities: implement the technique from the scratch or study its source code, if available, to adapt it to the new requirements of the new domain.

During the last ten years some generic solutions have been developed. They make easier the development of information visualization techniques. Some of these generalizations are: Piccolo [2], PREFUSE [3], or InfoVis Toolkit [4]. They cover an interesting body of techniques, such as visual and semantic zooming, zoom and pan, dynamic queries and magic lenses. But when a developer has to use one of these generalizations, the effort dedicated to learn their use is significantly high. We are interested in focus+context techniques [5], where we have not found any generalization for non Infovis experts.

Therefore, we are interested in assisting the interface programmers to use focus+context techniques with their own data. Thus, two issues have to be studied. Firstly, implications of the platform for which the generalization has been developed. And secondly, the independence from the nature of data handled and its adaptation, regarding the layout and manipulation in the workspace.

The rest of the paper is structured as follows. In section 2 we describe a specific focus+context technique called R-Zoom. Then, in section 3 we explain and evaluate its generalization. In section 4, starting from the previous process we propose a generalization framework. Finally, in section 5 we draw our conclusions.
2. R-ZOOM, A FOCUS+CONTEXT TECHNIQUE

R-Zoom is an information visualization technique designed to handle large scale workspaces with three special features: the elements are organized in a certain lineal order and both, a global and a detailed view of the elements is needed. Finally, R-Zoom tries to minimize location changes of elements.

This technique is founded on focus+context principles [5]. Also zoom and pan [2] capabilities were added. The main feature is that the user can increase the level of detail of any element inside the visible area without losing the context. We evaluated the usability of this technique in a comparative study against an overview+detail technique. R-Zoom obtained significantly better results than the overview+detail technique. For further details refer to [6].

2.1 Features of the elements

The elements are heterogeneous; the number of them is only limited by the hardware platform. R-Zoom will establish the level of detail used to represent the contents of each element. Users can change the level of detail applying it to all the elements or showing all the information about one specific element. Following the focus+context nomenclature, we will call focus to the element to which the highest level of detail is assigned, and context to the elements with the lowest level of detail. Usually, the level of detail will have an influence on the size of the element in the workspace.

2.2 The layout of the elements

As it has been previously mentioned, the elements have a sequential structure that must be kept. When all the elements are context ones –there is no focus– they are distributed by rows, from left to right inside each row, and from top to bottom of the workspace (see Figure 1a). Generally, the size of the focus and the context versions of an element are different. Therefore when an element is focused the distribution of all the elements will change. R-Zoom keeps the visual order minimizing the changes in the location of the elements.

When an element is selected to be the focus the following changes occur. The focus row is divided in two rows one with the context elements before the focus and another row with the elements next to the focus. One of the features of this technique is to keep locations of elements as much as possible; therefore the new focus will be placed in the gap of the first row (see Figure 1b). If there is not enough space in the first row, the focus is placed at the beginning of the second row (see Figure 1c). Finally if there is not enough space in both gaps, the biggest gap is selected and the level of detail of focus is adapted so it can fit in that gap.

Figure 1. Layout of the context elements: (a) without focus, (b) the focus is placed in the first row or (c) the focus is placed in the second row
3. RZ-UPDATE/RZ-COMPONENT, THE R-ZOOM GENERALIZATION

The main aim of this development is to extend the use of R-Zoom technique. For this reason we have decided to implement it as an API. In this section we detail the development of this API. Firstly it is necessary to choose the programming language which is going to be used. And secondly we have to study how to use language’s structures to build the API.

3.1 Programming language’s choice

We have considered the portability and the integration of the API in the structure of the programming language as main factors. We have decided Java is the ideal programming language to develop this API. On one hand, Java is platform independent, extending the use of the API. On the other hand, Java has a fixed and easily extendable user interface classes hierarchy.

3.1.1 Java’s hierarchy of graphical user interface classes.

In Java, the development of graphical user interfaces is based on three elements: components, containers and layouts. Components (objects from Component class) display the elements which can be use on an interface, ex. buttons, text boxes, forms, panels, etc. Some of these elements can have others inside, as forms or panels. These kinds of elements are containers (objects from Container class).

The element’s position in a container can be absolute, fixing each element its own position. However, generic positioning policies can be applied using layouts. A layout is encharged of elements’ positioning. Changing the layout is possible to change the position of the elements without having to modify them. Summarizing, the layout is assigned to the container; the components are added to the container and the layout specify how they are distributed in the container.

3.2 Division of the visualization responsibilities

In R-Zoom we can distinguish two aspects which have to be with visualization: the positioning of the elements on the window and the elements’ representation in focus/context version.

In order to maximize the technique’s generalization, we have decided that the elements of the workspace will be responsible of their own visualization. This means that the API’s users should develop this characteristic.

R-Zoom generalization consists of an API which implements a new layout. This layout communicates to the elements in the workspace: when they have to show their focus/context representation and where they have to do it. Elements give information about their size to the layout to make possible the distribution. Using a layout, we allow to use any kind of container, improving the technique’s generalization. Although the implementation of the elements is a responsibility of the user, it has to satisfy two conditions: to extend Java’s Component class so elements can be added to standard containers and to implement a simple interface so elements can communicate with the new layout. Following we describe the layout’s details and the interface to be implemented by the components.

3.3 Required features of the layout

Before developing a new layout we have checked the characteristics of the ones in Java. The requirements of our technique and its generalization are:

1. To work with components of different sizes.
2. To allow to specify the distance between elements in both dimensions.
3. To redistribute the elements when the size of the container changes.
4. To work with components which allow two different representations.
5. To give the user the necessary methods to determine the reduction factor of the contextual elements.
6. To be totally transparent to the user, allowing to use the layout as the others in the language.
7. To distinguish which element is selected as focus each moment.
8. To allow to inform the elements when they have to change between focus and context versions.
9. To redistribute the elements which do not change their visual representation when the focus is selected.
10. To allow different location policies of the focus depending on user’s preferences.

From the eight layouts offered by Java [7], logically all of them satisfied the transparency requirement (requirement 6), while working with different size components (requirement 1) is only satisfied by 4 of them. None of the layouts satisfied the rest of the requirements. According to this, we have to implement a new layout, which will be named RZ-Layout.

2.1 3.4 Required features of the component

The most important characteristic of the component is its functionality of changing the visual representation which is not influenced by any element. Besides, this element should be added to any container, independently of the layout which is being used.

As the components in Java have only one kind of visualization, we have to create a new type of component. It should be generic, allowing the visualization of any kind of element. These components consist of an interface that must be implemented by the user. As happens with the layout, the component will be named RZ-Component, taking into account the R-Zoom generalization.

3.5 The evaluation of the generalization

In order to evaluate this API we have measured the effort made by an user while using it in the development of a particular interface. We have measured the effort in terms of time dedicated to programming tasks.

The user was an expert programmer in the development of user interfaces with Java. The evaluation consisted of developing three interfaces which show different RZ-Component’s behaviors. During the implementation we measured times dedicated to: the use of the API, the implementation of the RZ-Component and the implementation of other aspects (menus, events, etc).

Next we describe each interface developed. All of them have a similar scheme: codification of a class which implements the RZ-Component interface and extends the Component class; the objects RZ-Components will be added to a container which will use RZ-Layout.

3.5.1 Interface 1: image visualization with R-Zoom style

This implementation consists of an application to visualize images, see Figure 4. The user can change the distance between rows, elements and the reduction factor applied to the images.

In this implementation, RZ-Layout is applied directly to the application window, avoiding the interaction with other elements of the language. The implementation of RZ-Component allows to display images, being necessary to keep both the original (the focus version) and the reduced one (the context version). The distortion used to generate the context version is implemented with Java’s methods to scale images.

Figure 4. A snapshot of the Interface 1, image visualization with R-Zoom style.
3.5.2 Interface 2: image visualization changing the container and the component

This implementation differs from the Interface 1 in: RZ-Layout applied to a container, the location of the focus and the improvement of the distortion used by the RZ-Component, see Figure 5.

In this case, we have applied the GridBagLayout to the application window. On the left will be a container with RZ-Layout where the focus is shown centered and on the right another with information about the selected component. The RZ-Component has the responsibility of show this information. Furthermore, we have improved the distortion applied, using alternative scaling methods.

Figure 5. A snapshot of the Interface 2, image visualization with Row-Centred style.

3.5.3 Interface 3: web pages visualization

This implementation allows to visualize a collection of web pages (see Figure 6). As in the Interface 1, we have applied the RZ-Layout to the application window.

This application connects to some given URLs given and for each one does parsing actions. The distortion applied to context elements is a semantic zoom of the web pages, which consists of a document with just: the page title, the background color, the first image and the h1, h2 and h3 headers.

Figure 6. A snapshot of the Interface 3, web pages visualization with overview+detail style.

3.5.4 Results

Table 2 summarizes the collected measurements during the development process: time dedicated to the use of RZ-Layout, time dedicated to the implementation of RZ-Component and time dedicated to other tasks.

As we can see in Table 2, time dedicated to the use of RZ-Layout is constant. However, time dedicated to the implementation of RZ-Component increases as the complexity in its implementation increases.
Furthermore, it can be appreciated that the time dedicated to the use of RZ-Layout in the Interface 3 is not significant in comparison with the total.

Table 2. For each example, effort dedicated to each task

<table>
<thead>
<tr>
<th>Total</th>
<th>Layout Building</th>
<th>Component impl.</th>
<th>Other tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface 1 2h</td>
<td>1.67%(2min)</td>
<td>30.83(37min)</td>
<td>67.5%(81min)</td>
</tr>
<tr>
<td>Interface 2 2:30h</td>
<td>1.33%(2min)</td>
<td>50%(75min)</td>
<td>48.67%(73min)</td>
</tr>
<tr>
<td>Interface 3 7:30h</td>
<td>0.45%(2min)</td>
<td>31.11%(140min)</td>
<td>68.44%(308min)</td>
</tr>
</tbody>
</table>

4. A FRAMEWORK FOR THE GENERALIZATION PROCESS

In this section, we reflect on the generalization process previously described. We propose a generalization framework for focus+context techniques. The main objectives of the generalization process are to make easier and amplify the use of the technique. First we study the factors to take into account from the point of view of the development of the generalization. Then, we analyze the generalization from the point of view of the nature of the visualized elements, its placement in the workspace and its manipulation.

4.1 The implementation platform

The development platform will state the programming language/environment that the users will use to develop their interface using the generalized technique. Firstly, we have to take into account the basic features: the extension possibilities of the development platform to receive the generalization and the performance requirements. Then we will have to study other features as the users’ required knowledge to use the generalization or how portable will be the technique developed using the generalization.

In case of the RZ-Layout/RZ-Component, choosing the Java programming language will not cause negative effects regarding the performance of the technique. In addition it is extensible due to its internal structure and highly portable. Finally, the users’ required knowledge to use the generalization is low because it is similar to the generic components used to develop user interfaces with Java.

4.2 Generalization in terms of the nature of the elements

The nature of the elements is twofold: individual and collective. The generalization of the individual nature causes the visual representation of the elements –context and focus version– to be independent from the technique. This happens with the RZ-Component elements, they generate their representation, reporting to the RZ-Layout on their size. An opposite example is the generic implementations of the TreeMaps technique [1]. Here, the individual nature of the elements is always the same, a number used to compute the size of the area.

When we refer to the collective nature, we are talking about the internal structure of the whole group of elements, their interrelations. Most of the focus+context techniques are strongly associated with the internal structure of the groups of elements (e.g. graphs [8], hierarchies [9] or tables [10]). The same occurs with RZ-Layout/RZ-Component. The internal structure of the groups of elements is ordered sequence. The generalization at this level would need to allow use the same technique with elements with different internal structure. We have not found any example for this kind of generalization.

In addition, regarding the nature of the elements, we have to take into account their continuity. It seems to be a feature of the internal structure, but it affects to the generalization of the individual nature, especially to the distortion applied to the context elements. Without continuity the distortion will be applied to each element, therefore the generalization at individual level does not change. But if the elements have continuity, aggregation could be considered as a feasible way to represent context elements. In this situation, the user...
will have to dedicate some effort to the development of the graphical representation of the context elements that actually are aggregated groups of elements.

4.3 Generalization in terms of the layout and the manipulation of the elements in workspace

The layout of the elements in the workspace is strongly related with the collective nature, e.g. tree organization of hierarchical structures, or the row based layout for ordered sequences. It is used to be a basic feature of the technique, and its generalization actually would be a parameterization of the layout of the elements, allowing to manipulate the basic features of the visual representation, e.g. sizes or distances.

The manipulation is also strongly associated with the family of techniques used. Thus, in case of RZ-Layout/RZ-Component we deal with a focus+context technique, therefore the basic actions will be focus and blur. Again the generalization is actually a parameterization of the way these actions are executed (pointing devices, cursor keys, etc). However, other possibilities can be taken into account. It can be allowed to graduate the distortion applied to the context elements, or include features from other interface families.

5. CONCLUSION

This paper is about the generalization of the focus+context interfaces. There are a lot of developments of this kind of interfaces, but we have not found any effort to generalize them.

Beginning with a particular technique developed for a specific domain, we have generalized it and obtained an API which includes the RZ-Component interface and the layout RZ-Layout. We have evaluated the generalization obtaining positive results, where the big effort is to develop the components, while time dedicated to the use of the layout is not significant. These results justify the design of the generalization, besides we can see how important is to generalize thinking in the development platform.

As a result of this process we propose a framework to generalize focus+context techniques. The most important factor in this process is the collective and individual nature of the elements in the workspace. Most of the focus+context visualization techniques are strongly related with the internal structure, the collective nature, of the groups of elements they manipulate. Thus, the corresponding generalizations actually are parameterizations of: the graphical properties of the technique –sizes, distances, etc– and the manipulation methods –pointing devices, cursor keys, etc–. Making generalizations independent of the individual nature results in powerful tools. Thus the users, who are not experts in information visualization, can use focus+context interfaces dedicating their development efforts to the representation of the elements, where they are experts.

We have identified two future lines of work. On one hand we will continue with the evaluation of the RZ-Component/RZ-Layout, considering users with different levels of expertise. On the other hand we plan to develop generalizations based on this framework of other well known focus+context techniques, e.g. the perspective wall [10] for a tabular domain or the hyperbolic browser [11] for a hierarchical domain.

ACKNOWLEDGEMENT

This project is supported by project TIN2008-04103/TSI of the Spanish Ministry of Science and Innovation.
REFERENCES


