Propuesta para la Generalización de Técnicas de Foco+Contexto

Francisco J. Almeida-Martínez, Jaime Urquiza-Fuentes,

J. Ángel Velázquez-Iturbide

Departamento de Lenguajes y Sistemas Informáticos I
Escuela Superior de Ingeniería Informática, Universidad Rey Juan Carlos
C/Tulipán s/n, 28933 Móstoles (Madrid), Spain
{francisco.almeida, jaime.urquiza, angel.velazquez}@urjc.es

Resumen. La manipulación de espacios de trabajo de gran escala es un problema al que se han dedicado muchos esfuerzos. Se han creado numerosas técnicas agrupadas en varias familias, casi todas con multitud de aplicaciones particulares e incluso soluciones más genéricas que facilitan el desarrollo de interfaces que usen dichas técnicas. Tan sólo la familia de técnicas de foco+contexto carece de este tipo de soluciones genéricas. Este trabajo expone una técnica particular de la familia foco+contexto llamada R-Zoom, así como su generalización. A partir de esta experiencia proponemos un marco de generalización que ayuda al planteamiento de generalizaciones de otras técnicas de la familia foco+contexto.

1 Introducción

La Visualización de Información es un campo de estudio que ha aportado numerosas soluciones para la manipulación de espacios a gran escala. Se han creado muchas técnicas que han probado su utilidad. Algunas de ellas incluso sacándolas del dominio original, p.ej. TreeMaps [1] y sus diferentes usos comerciales[1]. Cuando se aborda la utilización de una técnica en un dominio para el que no se ha implementado existen dos opciones, ambas bastante complejas, o se reimplementa la técnica, o se estudia el código ajeno -en caso de disponer de él- para adaptarlo a las necesidades.

A lo largo de la última década han aparecido generalizaciones que facilitan la implementación de diversas técnicas de visualización como Piccolo [2], PREFUSE [3], o InfoVis Toolkit [4]. Entre las tres cubren un conjunto interesante de técnicas como: zoom visual y semántico, zoom+desplazamiento, consultas dinámicas y lentes mágicas. Pero a la hora de usar este tipo de generalizaciones encontramos que el esfuerzo que hay que dedicar a su aprendizaje es bastante significativo[2]. Por otro lado, nuestro interés se centra en las técnicas de foco+contexto [5], para las que no hemos encontrado generalización alguna.

[1] http://www.smartmoney.com/marketmap/
http://www.hivegroup.com/gallery/home.html

[2] En algún caso incluso falta información necesaria, véase el manual de PREFUSE :
http://prefuse.org/doc/manual/interaction/dynamic queries/

Por tanto, la problemática radica en facilitar a un programador de interfaces gráficas la adaptación de técnicas foco+contexto a su propia información. Así, habrá que estudiar las implicaciones de la plataforma de implementación de la generalización, la independencia respecto a la naturaleza de la información que usará y su adaptación en cuanto a la disposición y manipulación en el espacio de trabajo.

El resto del trabajo se estructura como sigue. En la sección 2 describimos la implementación particular de una técnica de foco+contexto llamada R-Zoom. A continuación, en la sección 3 explicamos y evaluamos su generalización. La sección 4 propone un marco de generalización a partir del proceso anterior. Y finalmente, en la sección 5 describimos nuestras conclusiones.

2 R-Zoom, implementación particular de una técnica de foco+contexto

R-Zoom es una técnica de visualización de información diseñada para manipular espacios de trabajo de gran escala, donde los elementos están organizados según un cierto orden lineal, y se necesita tener una visión global de estos, a la vez que debe permitir una visión más detallada de alguno de ellos. Como característica añadida, R-Zoom tratará de minimizar los cambios producidos en la ubicación de los elementos.

Esta técnica se basa en los principios de las interfaces de *foco+contexto* [5], aunque de forma circunstancial se le añadió la posibilidad de manipular los elementos del espacio de trabajo mediante acciones de zoom+desplazamiento [2]. La característica principal es que dentro del área visible del espacio de trabajo se permite al usuario aumentar el grado de detalle de un elemento en concreto sin perder todo el contexto. Se evaluó la usabilidad de esta técnica comparándola con un enfoque de vista global+detalle, obteniendo unos resultados que favorecieron significativamente a R-Zoom. Se puede encontrar una descripción más detallada de la interfaz y su evaluación en [6].

2.1 Características de los elementos

Los elementos con los que trabaja R-Zoom podrán ser heterogéneos, no existiendo más límite para su número que el hardware sobre el que se ejecute la aplicación. Dichos elementos tienen una representación gráfica intrínseca. R-Zoom se encarga de fijar el grado de detalle con que se muestra su contenido. Con R-Zoom, el usuario podrá variar dicho grado de detalle, permitiendo elegirlo para su aplicación a los elementos, e incluso variarlo para mostrar toda la información necesaria de un elemento en un momento dado. Siguiendo la nomenclatura de las interfaces de foco+contexto, llamaremos foco al elemento mostrado con mayor grado de detalle y contexto a los elementos mostrados con menor grado de detalle. Normalmente, el grado de detalle influirá en el tamaño que el elemento ocupa en el espacio de trabajo.

2.2 Disposición de los elementos

Como hemos mencionado anteriormente, los elementos tienen entre sí una estructura secuencial que no puede variar. Cuando todos los elementos son contexto, R-Zoom los ubica en filas, de izquierda a derecha, colocando las filas de arriba abajo en el espacio de trabajo (Fig. 1a). Si uno de los elementos es el foco, y como éste generalmente ocupa más espacio que los elementos de contexto, la distribución de todos los elementos cambia, pero R-Zoom mantiene el orden visual y minimiza los cambios en la ubicación de los elementos. Así, suponiendo que todos los elementos de contexto tienen su posición según el eje de coordenadas cartesianas, R-Zoom mantiene las posiciones horizontales (eje X) de todos los elementos del contexto. De esta forma, mantenemos su orden secuencial y se muestra el foco lo más cercano a su posición en versión contexto.

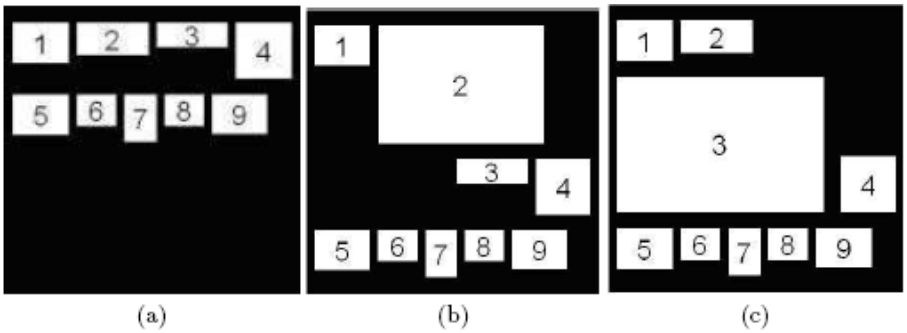

Fig. 1. Disposición de los elementos de contexto: (a) no hay ninguno enfocado, (b) el foco se sitúa en la primera fila o (c) el foco se sitúa en la segunda

El cambio producido por la existencia del foco consiste en dividir la fila a la que pertenece el elemento "enfocado" en dos, una fila con los elementos de contexto anteriores al foco y otra con los posteriores. Prioritariamente, el foco se coloca en el hueco de la primera fila (Fig 1b); si no hay espacio suficiente, se coloca en el hueco de la segunda (Fig. 1c). Finalmente, si no entra en ninguno de los huecos, se ubica en el mayor de los dos, ajustando el grado de detalle al espacio disponible.

3 RZ-Layout/RZ-Component, la generalización de R-Zoom

El objetivo principal de este desarrollo es la diversificación, en la medida de lo posible, del uso de la técnica R-Zoom. Por ello decidimos enfocarlo como el desarrollo de una API que permitiera su utilización. En primer lugar hay que elegir el lenguaje de programación a utilizar; en segundo lugar, estudiar cómo utilizar las estructuras de dicho lenguaje para el desarrollo de la API.

3.1 Elección del lenguaje de programación

Los factores principales que hemos contemplado para la elección del lenguaje de programación han sido la portabilidad y la integración de la API en la propia estructura del lenguaje.

Hemos decidido que Java es el lenguaje idóneo para el desarrollo de la API. Por un lado, Java proporciona independencia de plataforma, ampliando el ámbito de uso de la API. Por otro, Java posee una jerarquía de clases gráficas muy robusta y fácilmente extensible, como veremos a continuación.

Sobre la jerarquía de clases gráficas de Java. El desarrollo de interfaces gráficas en Java se basa en tres elementos fundamentales: componentes, contenedores y *layouts*.

Los componentes (objetos de la clase Component) representan los elementos que podemos utilizar en una interfaz, p.ej. botones, cajas de texto, formularios, paneles, etc. Algunos de estos elementos pueden albergar a otros en su interior, como son los formularios o los paneles. Este tipo de elementos son los contenedores (objetos de la clase Container).

La ubicación de los elementos dentro de un contenedor puede ser absoluta, fijando a cada elemento su posición, pero también se pueden establecer "políticas" de ubicación de los elementos, dando lugar a los *layouts*. Un layout es quien se encarga de ubicar los elementos de un contenedor, de forma que cambiando el layout se puede cambiar la ubicación de los elementos sin tener que modificarlos.

3.2 Reparto de responsabilidades sobre la visualización

En la descripción de R-Zoom podemos distinguir dos aspectos relacionados con la visualización: la disposición de los elementos en la ventana y la representación propia de los elementos en sus versiones de foco o contexto.

Para favorecer la generalización de la técnica hemos decidido que los elementos del espacio de trabajo sean quienes generen su propia representación visual. Esto implica que será el usuario de la API quien deba desarrollar esta característica. Por otro lado, la disposición de los elementos en el espacio de trabajo es la característica principal de la técnica, por lo que no tiene sentido trasladar esta responsabilidad a los elementos sino dejarla en la interfaz.

Por lo tanto, la generalización de R-Zoom consiste en una API que implementa un layout que comunica a los elementos del espacio de trabajo: cuándo deben mostrar su representación visual de foco o de contexto y dónde deben hacerlo. Los elementos informan al layout de su tamaño para que éste reparta el espacio de trabajo según su criterio. Al utilizar un layout, permitiremos usar cualquier tipo de contenedor, mejorando la generalización de la técnica. Aunque la implementación de los elementos sea responsabilidad del usuario, deben satisfacerse dos restricciones: para poder situar los elementos dentro de un contenedor deben heredar de la clase Component de Java y para poder comunicarse con el layout deberán implementar una interfaz[3] específica. A continuación describimos los detalles del layout, así como la interfaz que deberán implementar los componentes.

[3] Término de la orientación a objetos

3.3 Características del layout que se necesita

Antes de desarrollar un nuevo layout comprobamos las características de aquellos ofrecidos por Java. Los requisitos de nuestra técnica y su generalización son:

1. Trabajar con componentes de diferentes tamaños.
2. Permitir determinar la separación entre los elementos en ambos ejes.
3. Recolocar los elementos ante cambios del tamaño del contenedor manteniendo las distancias determinadas por el usuario.
4. Trabajar con un tipo de componente que pueda almacenar al menos dos tipos de aspectos.
5. Proporcionar los métodos necesarios para que el usuario pueda determinar el factor de reducción de los elementos contextuales.
6. Ofrecer total transparencia al usuario, pudiendo éste usar el layout de la misma manera que los que proporciona el lenguaje.
7. Distinguir qué elemento está seleccionado como foco en cada momento.
8. Permitir comunicar a un elemento cambios en su representación visual entre foco y contexto.
9. Recolocar los elementos que no cambian su representación visual cuando se cambie el foco.
10. Permitir variar la situación del foco en función de las preferencias del usuario.

De los ocho layouts ofrecidos por Java[4], obviamente todos cumplen el requisito de transparencia (requisito 6), mientras que el trabajo con componentes de diferente tamaño (requisito 1) sólo lo cumplen 4 de ellos y ningún layout cumple el resto de requisitos. Por lo tanto, deberemos implementar uno nuevo, que *llamaremos RZ-Layout.*

3.4 Características del componente que se necesita

La característica más relevante del componente es su capacidad para cambiar su representación visual y que este cambio no esté condicionado por ningún otro elemento. Además, este componente debe poder añadirse a cualquier contenedor, independientemente del layout que se le haya asignado.

Debido a que los componentes que proporciona Java sólo ofrecen un tipo de visualización, debemos crear un tipo de componente nuevo. Este componente debe ser genérico, es decir, debe permitir la visualización de cualquier tipo de elemento, por lo que se ha implementado como una interfaz[5]. Al igual que sucede con el layout, al tratarse de una generalización de R-Zoom, denominaremos al componente *RZ-Component*.

En el diagrama de clases UML resumido (Fig. 2) se ve cómo se han integrado ambos componentes. Por un lado tenemos las clases que permiten el funcionamiento

[4] http://java.sun.com/docs/books/tutorial/uiswing/layout/index.html
[5] Término de la orientación a objetos

correcto de RZ-Layout, y por otro están aquellas que dan soporte a RZ-Component. Tan sólo mencionar la clase RZ-ComponentInformado, interna a la API y totalmente transparente al usuario, ya que ayuda al mantenimiento de información relativa a los componentes necesarias para la API pero que el usuario no tiene porqué conocer.

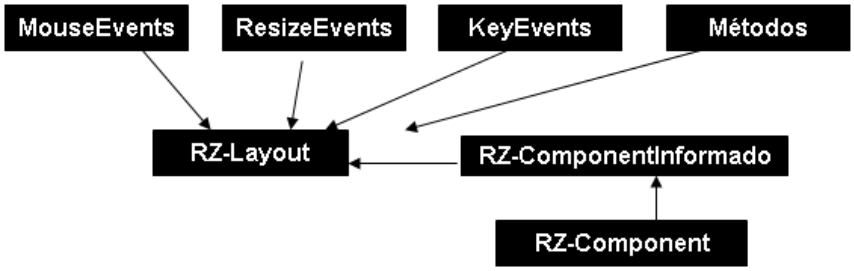

Fig. 2. Diagrama de clases UML resumido de la implementación de RZ-Layout/RZ-Component.

3.5 Evaluación de la generalización

Para evaluar esta API hemos comprobado el esfuerzo dedicado por el usuario a su utilización durante el desarrollo de una interfaz particular. Hemos medido el esfuerzo en términos del tiempo dedicado a programar.

El usuario fue un programador experto en desarrollo de interfaces. La evaluación consistió en el desarrollo de tres interfaces que muestran diferentes comportamientos de RZ-Component. Durante la implementación se midieron los tiempos dedicados a: la utilización de la API, la implementación de RZ-Component y a la implementación de otros aspectos (diseño, eventos, etc).

A continuación describimos brevemente cada interfaz desarrollada. Las tres tienen un esquema básico similar (Fig 3): codificación de una clase que implementa la interfaz RZ-Component y hereda de la clase Component; los objetos RZ-Component se añadirán a un contenedor que utilizará el layout RZ-Layout.

Interfaz 1: visualización simple de imágenes estilo R-Zoom. Esta implementación consiste en una aplicación para visualizar imágenes. Se permite que el usuario seleccione el directorio a visualizar, así como cambiar la separación entre filas, elementos y el factor de reducción aplicado a las imágenes.

En esta implementación RZ-Layout se asigna directamente a la ventana de la aplicación, evitando así que interactúe directamente con otros elementos del lenguaje. La implementación de RZ-Component permite visualizar imágenes, siendo necesario que se guarden la versión original y una reducida. Esta última, se obtiene utilizando los métodos de Java para obtener imágenes escaladas.

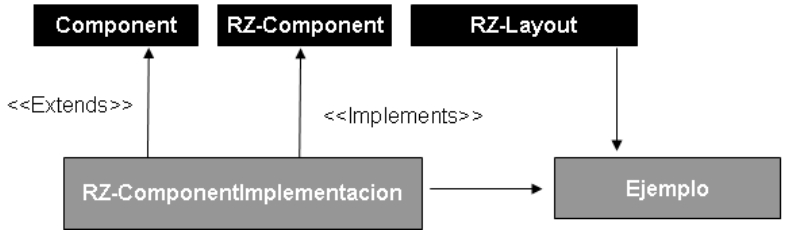

Fig. 3. Diagrama de clases UML resumido de la utilización RZ-Layout/RZ-Component. Las clases en fondo negro se proporcionan con la API, las de fondo gris dependen de la implementación del programador

Interfaz 2: visualización de imágenes variando el contenedor y el componente. Esta implementación difiere de la anterior en: asignación de RZ-Layout a un contenedor y mejora del RZ-Component.

En este caso se ha asignado a la ventana de aplicación el *GridBagLayout*. A la izquierda se va a colocar un contenedor con *RZ-Layout* y a la derecha otro con información sobre los componentes seleccionados. RZ-Component es el encargado de mostrar la información del elemento seleccionado. Por otro lado, se ha mejorado la forma de obtener las miniaturas, utilizando métodos alternativos a los lenguajes para obtener imágenes escaladas.

Interfaz 3: visualización de páginas Web. Esta implementación permite visualizar una colección dada de páginas web. Al igual que en el primer caso, se ha asignado RZ-Layout a la ventana de la aplicación.

Esta aplicación se conecta a varias URLs ya proporcionadas y para cada una de ellas realiza un parseo de los documentos web. Este proceso permite obtener la versión reducida de las páginas, que consiste en otro documento con parte del documento original. En concreto, la versión reducida consiste en: título de página, color de fondo, primera imagen y cabeceras h1, h2 y h3.

Resultados. Una vez que se han terminado cada una de las implementaciones y medido cada una de las variables (tiempo dedicado a la utilización de RZ-Layout, tiempo dedicado a la implementación de RZ-Component y tiempo dedicado a otras cosas), los resultados son los que se muestran en la tabla 1.

Como se puede ver en la tabla 1, el tiempo dedicado a la utilización de RZ-Layout es constante. Sin embargo, el tiempo dedicado a la implementación de RZ-Component va aumentando a medida que aumenta la complejidad en su implementación. Si nos centramos en el *Ejemplo 3,* se puede apreciar que el tiempo dedicado a la utilización de RZ-Layout es insignificante con respecto al total.

	Total	Creación Layout	Imp. Componente	Otras tareas
Ejemplo 1	2h	1.67%(2min)	30.83(37min)	67.5%(81min)
Ejemplo 2	2:30h	1.33%(2min)	50%(75min)	48.67%(73min)
Ejemplo 3	7:30h	0.45%(2min)	31.11%(140min)	68.44%(308min)

Table 1. Tabla de esfuerzo dedicado a cada tarea en cada ejemplo implementado

4 Proceso de generalización

En esta sección reflexionamos sobre el proceso de generalización descrito anteriormente y proponemos un marco de generalización de técnicas de foco+contexto. El objetivo fundamental de la generalización es facilitar y ampliar el uso de la técnica implementada. En primer lugar estudiamos las cuestiones a tener en cuenta desde el punto de vista de la implementación de la propia generalización. A continuación, analizamos las posibilidades de generalización desde el punto de vista de la naturaleza de los elementos visualizados, su disposición en el área de trabajo y su manipulación.

4.1 Plataforma de implementación

La plataforma de implementación determinará el lenguaje/entorno que un usuario utilizará para implementar la interfaz que use la técnica generalizada. En primer lugar hay que tener en cuenta cuestiones básicas, como la extensibilidad de la plataforma para acoger la generalización o los requisitos de rendimiento. Una vez contempladas las cuestiones básicas, se deberán estudiar otras, como el conocimiento necesario por parte del usuario que vaya a utilizar la generalización o la portabilidad de la técnica desarrollada con la generalización.

En el caso de RZ-Layout/RZ-Component, la elección del lenguaje de programación Java no conllevará penalizaciones serias en cuanto a rendimiento mientras que es fácilmente extensible gracias a su estructura interna y altamente portable, ya que es multiplataforma. Finalmente, el conocimiento necesario por parte del usuario para usar la generalización es poco, ya que su uso es muy similar al uso de los componentes genéricos usados en cualquier interfaz desarrollada en Java.

4.2 Generalización según la naturaleza de los elementos

La naturaleza de los elementos tiene dos vertientes: individual y colectiva. La generalización de la *naturaleza individual* hace que la representación visual –en forma de foco o contexto- de cada elemento, como unidad aislada, sea independiente de la técnica. Esto ocurre con los *RZ-Component*, ya que son ellos quienes generan totalmente su representación visual informando a *RZ-Layout* sobre el espacio que ocupan, para que éste pueda colocarlos como mejor convenga. Un ejemplo contrario son las implementaciones genéricas de la técnica TreeMaps [1]. En este caso la

naturaleza individual de los elementos es la misma, una cantidad que junto con otra información se usará para calcular la superficie que ha de ocupar.

Cuando hablamos de *naturaleza colectiva* nos referimos a la estructura interna del conjunto de los elementos, sus interrelaciones. La mayoría de técnicas foco+contexto desarrolladas están fuertemente relacionadas conla estructura interna de los elementos (p.ej grafos [7], jerarquías [8] o tablas [9]). Esto mismo ocurre con *RZ-Layout/RZ-Component* cuya estructura interna son secuencias ordenadas de elementos. La generalización a este nivel implicaría permitir usar la misma técnica con elementos estructurados de distinta forma, no hemos encontrado ejemplos de este tipo de generalización.

Además de estas dos vertientes de la naturaleza de los elementos, hay que tener en cuenta su continuidad. Aunque en principio es una característica de la estructura interna, también afecta a la generalización a nivel individual y en concreto a las distorsiones aplicadas a los elementos del contexto. Si no hay continuidad, la distorsión aplicada afecta únicamente a cada elemento, de forma que la generalización a nivel individual no cambia. Si hubiera continuidad se podría considerar la posibilidad de "resumir" conjuntos de elementos en uno nuevo que los represente a todos. En esta situación, el usuario debería hacer un esfuerzo de implementación para tener en cuenta a los elementos de contexto que representan a conjuntos de elementos.

4.3 Generalización según la disposición y la manipulación de los elementos en el espacio de trabajo

La disposición de los elementos en el espacio de trabajo se relaciona fuertemente con la naturaleza colectiva, p.ej. la disposición arbórea de las estructuras jerárquicas, o en nuestro caso la disposición lineal en filas por ser una secuencia. Suele ser una característica básica de la técnica y su generalización sería más bien una parametrización de la disposición de los elementos, permitiendo manipular las características básicas de la representación visual como distancias o tamaños.

La manipulación también depende en gran medida de la familia de técnicas con que se trabaja. Así en el caso de RZ-Layout/RZ-Component hablamos de una técnica foco+contexto, por lo que las acciones básicas serán enfocar y desenfocar. De nuevo la generalización termina siendo una parametrización del modo de ejecutar dichas acciones (dispositivos apuntadores, teclas de cursor, etc). Sin embargo, se pueden contemplar otras posibilidades como permitir graduar la distorsión de los elementos de contexto o incluir propiedades de otras familias de interfaces.

5 Conclusiones

Este trabajo trata sobre la generalización de interfaces foco+contexto. Existen gran cantidad de desarrollos de este tipo de interfaces, pero no hemos podido encontrar ningún esfuerzo conducente a su generalización.

Partiendo de una técnica particular desarrollada para un dominio concreto, se ha generalizado obteniendo un API[6] que incluye la interfaz RZ-Component y el layout RZ-Layout. Evaluamos dicha generalización obteniendo unos resultados claramente

[6] http://www.lite.etsii.urjc.es/rzlayout/index.html

positivos, donde el esfuerzo principal se dedica al desarrollo de los componentes, mientras que el esfuerzo dedicado al uso del layout es mínimo. Esto justifica la implementación de la generalización, además se ve claramente cómo el hecho de generalizar pensando en la plataforma de implementación permite aprovechar al máximo el conocimiento del usuario sobre el desarrollo de interfaces.

Como resultado de este proceso proponemos un marco para la generalización de técnicas de foco+contexto. El mayor peso en este marco de generalización depende de la naturaleza de los elementos del espacio de trabajo. Independizando la técnica de la naturaleza de los elementos conseguimos ampliar el uso de las técnicas de foco+contexto a usuarios no expertos en visualización de información. El diseño de técnicas de foco+contexto se centra en la disposición de los elementos en el espacio de trabajo así como su manipulación, por ello su generalización se materializa en parametrizar las características de la técnica.

Referencias

1. Johnson, B., Shneiderman, B.: Tree-maps: A spacelling to the visualization of hierarchical information structures. In Card, S., Mackinlay, J., Shneiderman, B., eds.: Information Visualization: Using Vision to Think. Morgan Kaufmann Publishers (1999) 152-159
2. Bederson, B., Grosjean, J., Meyer, J.: Toolkit design for interactive structured graphics. IEEE Transactions on Software Engineering 30(8) (2004) 535-546
3. Heer, J., Card, S.K., Landay, J.A.: Prefuse: A toolkit for interactive information visualization. In: CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, New York, NY, USA, ACM (2005) 421-430
4. Fekete, J.D.: The infovis toolkit. In: IEEE Symposium on Information Visualization (INFOVIS'04), Los Alamitos, CA, USA, IEEE Computer Society (2004) 167-174
5. Card, S., Mackinlay, J., Shneidermann, B., eds.: Readings in Information Visualization: Using Vision to Think. Morgan Kaufmann Publishers (1999)
6. Urquiza-Fuentes, J., Velázquez-Iturbide, J., Lázaro-Carrascosa, C.: Design and evaluation of R-Zoom, a new focus+context visualization technique. In: INTERACCIÓN 2006 Diseño de la Interacción Persona-Ordenador: Tendencias y Desafíos, Ciudad Real, España, Universidad de Castilla la Mancha (2006) 91-100
7. Sarkar, M., Brown, M.: Graphical ¯sheye views. Communications of the ACM 37(12) (1994) 73-84
8. Robertson, G., Mackinlay, J., Card, S.: Cone trees: Animated 3d visualizations of hierarchical information. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems '91 (CHI '91), New York, NY, USA, ACM Press (1991) 189-194
9. Mackinlay, J., Robertson, G., Card, S.: The perspective wall: Detail and context smoothly integrated. In: Conference proceedings on Human Factors in Computing Systems '91 (CHI '91), New York, NY, USA, ACM Press (November 1991) 173-179