A Toolkit for Addressing HCI Issues in Visual Language Environments

Emmanuel Pietriga
INRIA Futurs & Laboratoire de Recherche en Informatique (LRI)
Bât. 490 - Université Paris Sud, 91405 Orsay, France
emmanuel.pietriga@inria.fr

Abstract

As noted almost a decade ago, HCI (Human-Computer Interaction) aspects of visual language environments are under-developed. This remains a fact, in spite of the central role played by user interfaces in the acceptance and usability of visual languages. We introduce ZVTM, a toolkit aimed at promoting the development of HCI aspects of visual environments by making the creation of interactive structured graphical editors easier, while favoring the rapid integration of novel interaction techniques such as zoomable user interfaces, distortion lenses, superimposed layers, and alternate scrolling and pointing methods.

1. Introduction

Visual language environments ease the programmer's task and increase his productivity by providing him with debugging tools as well as syntax- and semantics-aware editing primitives. Combined with a high level of *liveness*, these primitives limit the level of *viscosity* associated with the manipulation of visual language programming constructs, and can prevent the programmer from making some kinds of errors [20]. Other representation and interaction issues such as the screen real-estate problem are also addressed at the level of the visual environment, which therefore plays a central role in the usability of visual languages and their acceptance by programmers [14].

The implementation of visual language environments can be achieved through the use of various visual language meta-tools (see [27] for an example and list of references). It can also be done from scratch, coding the user interface with a combination of high-level WIMP *(Window Icon Menu Pointer)* toolkits (e.g. Java/Swing) and lower-level graphical APIs (e.g. Java2D) which typically provide methods for drawing shapes and manage low-level events through a set of callbacks. They are used to build application-specific interface components in which arbitrary graphical objects can play the role of widgets and are necessary for instance to represent visual programming language constructs or any other complex representation that cannot be handled by traditional WIMP widgets.

Low-level graphical APIs are powerful but cost more implementation and maintenance time, and put more burden on the programmer of the visual environment. As a consequence, and because novel interaction techniques are themselves difficult to implement, HCI aspects of visual language environments and visual language meta-tools are often under-developed as noted by Green almost a decade ago [20]. Such aspects are nevertheless important and deserve more attention and effort on the part of visual language environment designers and implementors.

One way to promote the integration of new interaction techniques in visual environments is to build toolkits that reduce the cost of implementing the user interface part of interactive software while offering off-the-shelf customizable interaction components, as considered in [2]. In this paper we describe ZVTM (*Zoomable Visual Transformation Machine*), a Java-based toolkit designed to ease the task of creating the complex interface components required by visual language environments, visual language meta-tools or graph editors, while favoring the rapid integration of novel interaction techniques from which these applications can significantly benefit.

1.1. Related work

CPN2000 [3] is a good but rare example of use of novel interaction techniques in a visual editor, experimenting with toolglasses, bi-manual manipulation and marking menus for colored Petri net editing. Few visual language environments and meta-tools make use of such techniques in their user interface. One notable exception is Pounamu, which provides ZUI *(Zoomable User Interface)*-based multiple consistent views to address the screen real-estate problem [25].

Pounamu's zoomable views are implemented with Jazz. Along with its replacement Piccolo [4], this ZUI toolkit is similar in its purpose to ZVTM. But our approach relies on a more human-centric representation model and tries to

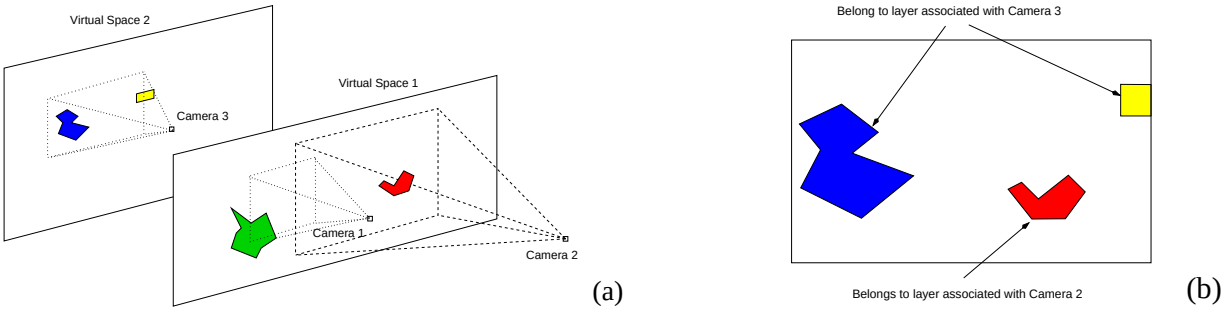

Figure 1. Schematic representation of virtual spaces, cameras, glyphs, views and layers

be more lightweight. While acknowledging the richness of scene graphs, we chose to use a more basic approach, taking into consideration the fact that applications often have to maintain their own internal representation of what is being specified (e.g. the structure of a visual program). Having to maintain a separate scene graph for the graphical representation puts significant programming burden on the application programmer and consumes more memory than a simple stack of (possibly linked) graphical objects.

Earlier toolkits [13, 22, 26] implement specific interaction techniques. ZVTM gathers many of these in a single extensible framework, favoring their combination through a simple API. In this respect it is closer to information visualization toolkits [19, 21]. These, however, provide dedicated data structures and high-level manipulation methods optimized for the visualization of very large quantities of information, and are not aimed at building highly customized applications such as visual language environments.

1.2. Applications

ZVTM is distributed [18] under an open source license and can be used in both free and commercial products. To our knowledge, it has been used in a dozen different applications, among which four have been designed and implemented by the author. These applications range from visual language environments to distributed system monitoring facilities and domain-specific graph editors. After giving an overview of the toolkit, we discuss common issues associated with user interface design and implementation of visual environments, relying mainly on the following two applications to illustrate how they were addressed with our toolkit.

VXT [30] is a domain-specific visual programming language for the specification of structural XML transformations. It is a declarative rule-based language with some control structures, relying on a unified treemap-based visual formalism [37, 30] for representing document structures, document schemas and transformation rules. The eponymous visual environment was the first application based on ZVTM and uses many features of the toolkit.

IsaViz [17] is a visual authoring tool for RDF [38] designed and distributed by the World Wide Web Consortium (W3C). RDF models are graphs whose textual serializations in RDF/XML or other triple-oriented formats are not user-friendly, partly because they fail to convey the models' graph structure. IsaViz generates editable visual representations as zoomable 2D graphs which are often easier to understand. While not a visual language environment, similar representation and interaction issues arise, as IsaViz is a visual editor aware of the structure and semantics of the data being manipulated.

2. Overview

ZVTM provides the application programmer with building blocks for implementing complex 2.5D (zoomable) interface components that cannot be handled by traditional WIMP widgets. It also features off-the-shelf visualization and navigation components that can easily be combined.

2.1. Concepts and architecture

The toolkit is based on the metaphor of infinite[1] universes called *virtual spaces* that can be observed through movable and zoomable *cameras* and contain potentially large amounts of geometrical shapes called *glyphs*.

All glyphs rely on the same polymorphic object model. A glyph belongs to a specific virtual space, but can be observed through different cameras simultaneously as each virtual space can contain multiple cameras as shown in Figure 1-a. Cameras are associated with viewports called *views* which correspond to windows in the user interface. Distortion lenses [12] can be applied to views. If more than one camera is associated with a given view, each camera draws its content on a transparent layer (Figure 1-b). The glyph graphical model features alpha channel support; glyphs can therefore be opaque, translucent or transparent.

[1] Actually bounded by the representation of 64-bit integers in the current Java implementation, i.e. $\pm 9 \times 10^{18}$.

Translucency is one of several orthogonal visual variables that define a glyph. Modifications to these variables can be animated using various temporal schemes (see section 5). Camera translations and altitude changes can also be animated, as well as distortion lens modifications.

Input event handling is managed through high-level callbacks that provide context about the event such as the list of objects intersected by the cursor. The event handler's interface can be extended at will to support non-standard events, such as the use of two pointing devices simultaneously (see section 6).

On top of these concepts are implemented various interaction techniques that can be combined: zoomable user interfaces, superimposed translucent layers, fisheye lenses, rate-based scrolling and speed-dependent automatic zooming, semantic pointing, and more. The use of these techniques is illustrated in the next sections with the two applications described earlier.

2.2. Low-level graphical operation management

Toolkits such as Java2D are powerful but difficult to use, requiring the programmer to deal with low-level graphical operations and implementation problems. ZVTM allows the programmer to consider the task at a more abstract level by automatically handling the following operations:

Clipping: the toolkit is designed to handle many glyphs while maintaining an acceptable refresh rate. Two ZUI-aware clipping algorithms contribute to this: an analytical clipping pass that determines whether each glyph should be projected and painted or not (based on its bounding box in virtual space), and an optional top-down pass that can detect some occlusion configurations and ignore glyphs that will not be visible in the final rendering (taking glyph translucency into account).

Multi-threading and consistency: multiple cameras associated with different views can exist simultaneously, observing the same or different regions of (possibly different) virtual space(s). A glyph observed through different cameras simultaneously is actually a single object with multiple projected coordinates. As a consequence, the synchronization and consistency of multiple views is automatic.

Repaint requests: views are refreshed lazily, i.e., only upon request. Such repaint requests are fired automatically, so that the programmer only has to assign new values to glyph and camera visual properties. Each camera is associated with a view which runs in its own thread. Each thread tries to provide a frame rate as close to 25 images per second as possible (unless specified otherwise), but not beyond this limit as it would consume CPU resources for nothing.

Animation management: all camera, glyph and lens animations are handled by the same module, which offers a simple and unified API for their declarative specification.

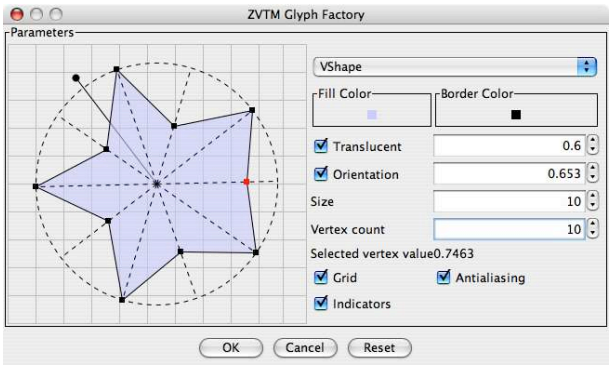

Figure 2. VShape model

The module manages a queue of pending animations and can handle concurrent animations affecting orthogonal visual variables of the same object.

Hardware acceleration: various graphics acceleration methods are available, such as Java Volatile Images and various OpenGL rendering pipelines. ZVTM features different types of views that provide hardware acceleration based on these methods while sharing the exact same API.

These mechanisms make the user interface part of the application less difficult and less time-consuming to implement, thus allowing the programmer to devote more time to enhancing the interface's quality, features and design.

3. Mapping data to visual variables

As stated in [20], "*data must be presented in a usable form before it becomes information, and the choice of representation affects the usability*". The representation system, and thus the graphical object model, play a central role in converting data into information. Low-level graphical APIs provide large sets of powerful drawing primitives that address many programmers' and designers' needs. However, these primitives are often specific to the associated geometrical shape, and APIs suffer from the lack of a unified set of instructions for manipulating heterogeneous graphical objects. Moreover, these instructions often rely on machine-oriented models for encoding geometrical shapes. Such models have advantages (e.g. performance) but do not make the mapping of data to visual variables straightforward.

ZVTM's graphical object model is inspired by Bertin's perceptual dimensions [5] and the Visual Abstract Machine's visual type system [36]. The model uses encapsulation to provide the programmer with a polymorphic instruction set for manipulating all graphical objects, called glyphs, no matter their actual shape and appearance[2].

[2] In this respect we diverge from the VAM's visual type system which is intrinsically polymorphic but limited in the kind of shapes it can represent, as ellipses, rectangles, bitmap images and splines are not supported.

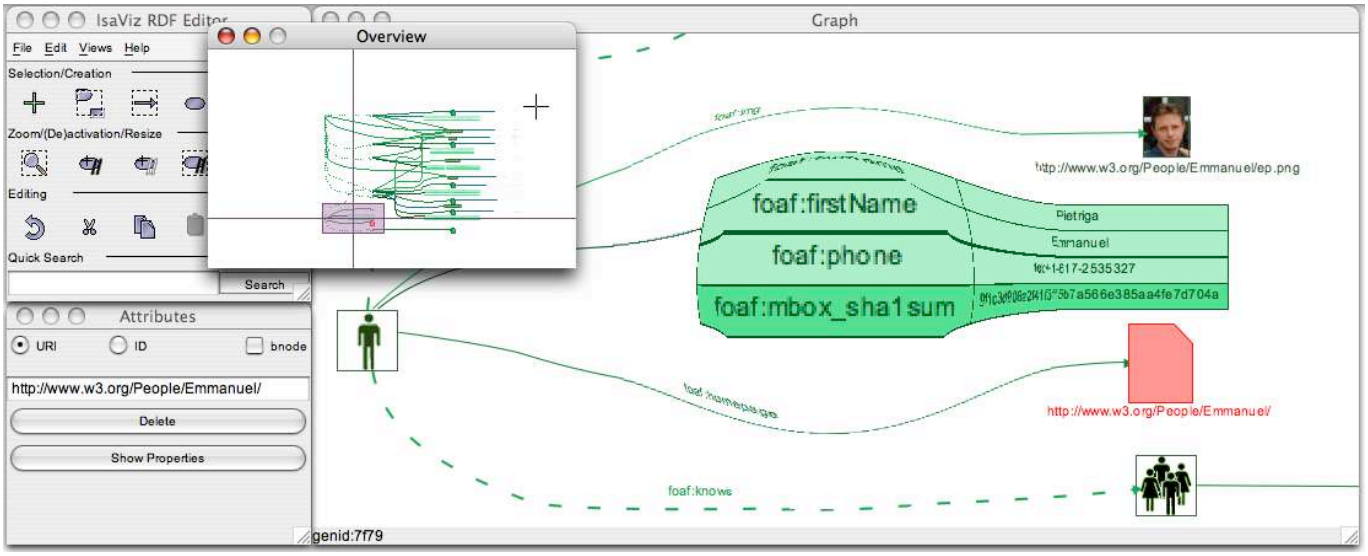

Figure 3. GSS rendering of an RDF graph in IsaViz, using a ZUI, a distortion lens and an overview

All glyphs are defined by the following orthogonal attributes (Figure 2): the cartesian coordinates of the shape's centroid, the size of the shape's bounding circle, the shape's orientation, its border and fill colors, associated with an optional alpha channel for translucency. Basic predefined shapes are fully defined by these attributes. Other, more complex, shapes may require additional attributes. For instance, *VShape* glyphs support an arbitrary number of vertices, whose position within the bounding circle is represented by a normalized float (see [36] for more details). Glyphs can also be composed of other glyphs and still define polymorphic operations (resizing, reorienting, translating, coloring).

The resulting representation system, with its orthogonal visual variables that mirror perceptual dimensions, makes mapping data to graphical attributes easy. It is coupled with the direct manipulation interface shown in Figure 2 that facilitates the definition of glyphs, drawing a parallel with [16] which demonstrates the importance, for color selection, of a well-designed interface over the supposed intuitiveness of color models.

Both GSS [29], the Graph Stylesheet language for RDF graphs implemented in IsaViz (Figure 3) and VXT (Figure 4) rely heavily on the glyph model to dynamically map data and represented object properties to graphical attributes in a straightforward manner.

4. Scalability of representations

One problem commonly associated with visual programming languages is the diffuseness of notations and the consequent need for screen real-estate (a limited resource). Both representation and interaction solutions can be explored to address this problem. For instance, designing domain-specific languages tends to increase the *closeness of mapping* [20], which is often accompanied by a reduction of the number of lexemes required to express a given concept or instruction. However, this is very dependent on the language and underlying paradigm. Generic solutions to the problem are more likely to be found on the side of HCI aspects of the interface. ZVTM makes available various techniques coming from the field of information visualization aimed at addressing scalability issues.

4.1. Focus+context techniques

Visualizing large quantities of information poses the problem of getting a detailed view of the current area of interest while maintaining user awareness of the global structure. It is also important to put the focus area in context in order to avoid user disorientation. This can be achieved by different means such as distortion lenses which provide a smooth transition between the focus and context, overviews (also called radar views) identifying the current focus area, or superimposed layers.

In order to support the display of large RDF graph structures, IsaViz uses a ZUI coupled with geographic bookmarks and a set of navigation helpers to quickly navigate in the representation, allowing smooth transitions from global to detailed representations of (areas of) the graph. However, these facilities by themselves are insufficient for very large graphs, as it is still easy to get lost in the representation.

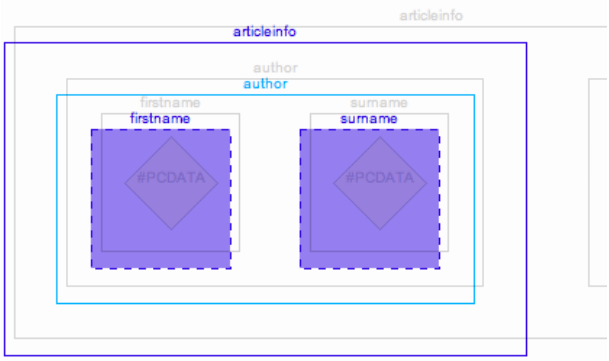

Figure 4. *VPME* represented as a visual filter above the source structure in VXT (two layers)

Two focus+context techniques are used to further assist the user (Figure 3):

- an additional ZVTM view displaying an overview of the graph and identifying the current region observed through the main camera,
- a distortion lens providing a detail-in-context representation of the region of interest.

Thanks to the multiple synchronized views mechanism, implementing the overview required less than twenty lines of code, event callbacks for controlling the main camera from the overview window included. Activating or deactivating the distortion lens required only one line of code.

Lenses are based on Carpendale and Montagnese's framework [12]. Right now, the following drop-off functions are available: Linear, Inverse Cosine and Manhattan. They can be used in conjunction with the following distance functions: $L(1)$, $L(2)$, $L(\infty)$. Other functions can easily be added to the toolkit simply by subclassing existing lenses and overriding the mathematical functions defining the drop-off profile and L_p-metric. All lens-related operations, including animation capabilities, are automatically available for every lens type.

Another focus+context technique, which also addresses the screen real-estate problem, is to display different observation regions on superimposed translucent layers, which thus share the same physical screen space. As mentioned in section 2.1, ZVTM supports the combination of different cameras in the same view, organized in layers. Used in conjunction with translucent objects, this mechanism makes the creation of overview layers [15, 24] straightforward. As in [31], such layers can be displayed temporarily. Layers can also be used for different purposes. A separate layer can for instance store toolglasses (see section 6), or be used in a visual programming environment to display secondary notations such as comments referring to visual source code, which are then easy to separate from the actual programming constructs.

Layers can also be used in more innovative ways. For instance, the visual interface of Blackwell's SWYN [7] for constructing regular expressions from examples could easily be implemented in ZVTM. In a related application domain, VXT uses superimposed layers to represent instances of XML documents or schemas in the background, while the foreground layer contains *Visual Pattern Matching Expressions (VPMEs)* that select source nodes and extract information from the source structure (they are the left-hand side of transformation rules). Taking advantage of VXT's unified visual formalism based on treemaps, this multi-layer representation conveys the idea of visual filters, as a *VPME* closely resembles the subtrees it matches from a purely visual perspective (Figure 4). The VXT environment thus offers an intuitive metaphor for evaluating pattern matching expressions against source XML structures, exploiting human visual capabilities and allowing programmers to reason about transformation rules visually. To address the problem of visual interference caused by their superimposition, the two layers are visually differentiated by rendering the background source structure using shades of gray with minimum contrast while *VPMEs* are rendered with highly-contrasted vivid colors, following Tufte's minimal contrast principle [35].

4.2. Semantic zooming

In the previous section we presented techniques to address the problem of providing detailed in-context views of large-workspace regions on a limited display surface. These techniques all rely on geometrical transformations of graphical objects, either through magnification or distortion, but do not tamper with the object's appearance and content. A complementary approach implemented by ZUI toolkits is to modify the appearance of objects as the amount of screen real-estate available to them changes. This technique, called *semantic zooming* [28], makes it possible to render objects with varying levels of details depending on the camera's altitude of observation.

As semantic zooming is a domain-specific issue, ZVTM does not provide an "implementation" of it, as no such thing exists, but supports it through the definition of new glyph types. The amount of screen real-estate available for the rendering of a glyph instance (i.e. its apparent size) is known by its rendering method through the projection process that occurs earlier in the display pipeline. Therefore, programmers willing to create a glyph type that exploits semantic zooming only have to subclass an existing glyph type and override the rendering method. The apparent size value computed through projection by the ZVTM engine can then be used to decide what level of details to select.

In its implementation of the second version of the GSS stylesheet language for RDF [29] (currently under development), IsaViz makes use of semantic zooming to enable the display of a varying number of Web resource properties depending on the altitude of observation.

4.3. Procedural abstraction

Scalability of a visual programming language is also tied to the possibility of using procedural abstractions to encapsulate subtask details [9]. Using such abstractions, the complexity of the visualization can be managed by splitting programs' representations into different views (see e.g. LabView's *virtual instruments*). Through the use of multiple virtual spaces, cameras and views, ZVTM facilitates the implementation of procedural abstractions independently of the programming paradigm, and provides building blocks for addressing visibility and juxtaposability issues, i.e. the ability to see different parts of the code simultaneously [20].

As an example, RDQLPlus[3], a visual tool based on ZVTM for building RDQL queries, uses multiple virtual spaces and views to store and display separate queries, thus making their comparison easier. Another example of procedural abstraction achieved through the use of ZVTM components is VXT's environment. The multilayer representation described in section 4.1 is part of a global effort to find new ways of organizing the visual programming environment. Instead of splitting the workspace on the basis of the heterogeneity of entities manipulated by the programmer (source XML structure, schema, rules), we explored a more task-centric approach while keeping scalability problems in mind. The main window contains the above-mentioned source and *VPME* layers and is thus dedicated to the task of selection and extraction of source data. New windows focusing on the transformation task performed by each rule can be opened simultaneously and closed at will, taking advantage of rule independence from the perspective of their specification and modification by VXT programmers.

5. Perceptual continuity

Sudden and unexpected changes in a user interface (e.g. objects moving from one position to another without any transition between the two states) put a heavy cognitive load on the user, who must mentally relate the states in order to re-assimilate the new display [13]. Animating changes applied to user interface objects transfers part of the user effort to the perceptual level, freeing cognitive processing capacity for application tasks [32]. Aside from the aesthetically pleasant impression it produces on most users when used appropriately, animation therefore contributes to make interfaces more user-friendly.

[3] http://rdqlplus.sourceforge.net

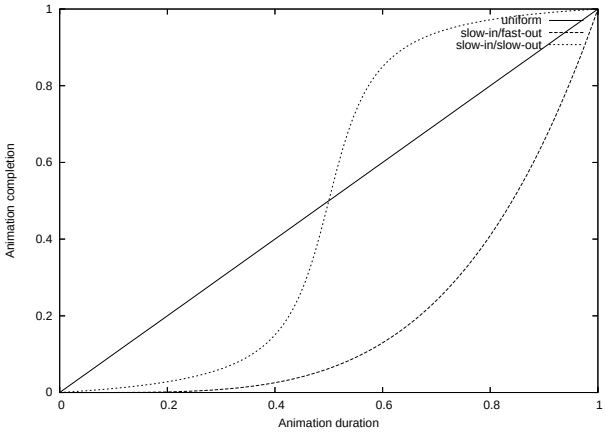

Figure 5. Animation pacing functions

Animations have been used for didactic purposes, for instance to explain algorithms (see e.g. [11, 33]), but tend to be used more and more just for their above-mentioned ability to reduce the user's cognitive load. For instance, desktop animations are ubiquitous in Apple's Mac OS X and contribute to the perceived quality of this operating system's GUI. Closer to our domain, many information visualization toolkits also support some animation primitives often centered on objects' position. Other toolkits [13, 22] provide more advanced animation support, but the use of animation in visual language environments and other visual editors remains limited.

ZVTM offers animation capabilities inspired by Stasko's path/transition paradigm [34]. Many user interface changes can be animated following a unified declarative API. Variables to which animations can be applied include all basic glyph variables (position, orientation, size, color and translucency) as well as some shape specific ones (bezier curve control points). Camera translations and altitude changes (zoom-in/out) can also be animated, as well as distortion lens' radii and magnification factor modifications. Animations are specified with a single instruction and do not require more implementation work than basic variable value modifications. They require the following parameters: the animation duration, the object involved, the variable(s) impacted by the animation, the desired target value interpreted as an offset from the start value, and the pacing function. Three pacing functions are available (Figures 5 and 6). Slow-in/slow-out transitions are typically used for camera motion (e.g. in IsaViz) and some glyph animations such as translations, as they convey a feeling of solidity that is important in direct manipulation interfaces [13]. Non-uniform pacing functions are generally used to put emphasis on the start (and end) of animation paths. However, they are not always appropriate, and the uniform function is often used when modifying distortion lens parameters or animat-

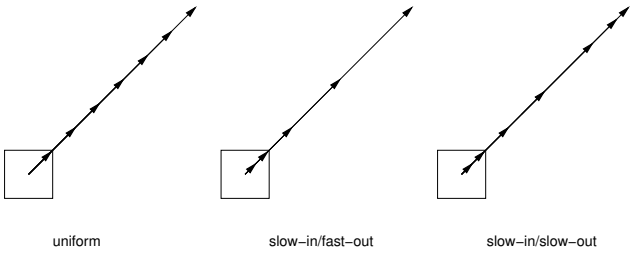

Figure 6. Pacing functions for a translation

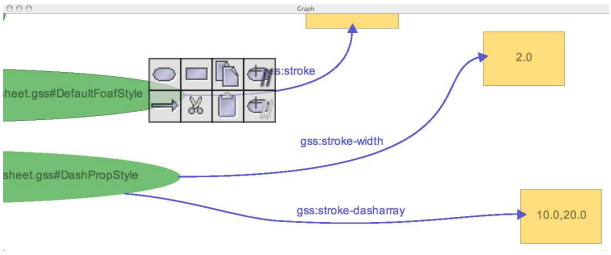

Figure 7. Click-through tools in IsaViz

ing color changes (e.g. when applying the minimal contrast principle in VXT's superimposed layers - see section 4.1).

Animations are managed by a dedicated thread that controls their timing precisely, skipping steps on the computed animation path if necessary. This mechanism ensures that an animation lasts the specified duration, and runs as smoothly as the system and hardware performance allow.

6. User input

So far we have discussed issues related to the representation of information and its perception by users. Another fundamental aspect of interaction covers the means by which the user operates the environment through various input devices and interface components. Many novel techniques have been developed that try to address the limits of traditional WIMP interfaces. However, few are widely used because of the users' cost of change and the lack of associated models and tools for their implementation [2]. ZVTM's constructs and event handling mechanism can be used to quickly implement the following techniques.

Speed-dependent automatic zooming [23] affects camera altitude in order to maintain a constant perceptual scrolling speed in screen space. This technique builds on the rate-based scrolling technique commonly used to navigate ZVTM unbounded spaces. Its implementation in ZGRViewer[4] (a visualization tool for Graphviz[5]) only required a dozen lines of code.

Bi-manual interaction [10] exploits the human ability to perform separate tasks with each hand in order to improve interaction performance. ZVTM allows two pointing devices to be used simultaneously (only under Linux with add-on module ZVTM-MPD until the USB protocol gets supported by Java on other platforms). For instance, a mouse can be used by the dominant hand for tool manipulations and camera translations while the non-dominant hand uses a trackball to control camera altitude.

Toolglasses [6] are used in conjunction with bi-manual interaction to provide the user with a translucent palette containing click-through tools. Such widgets do not require dedicated screen space and are useful for tasks that require switching between tools repeatedly, as frequent shuttling between the tool palette and the drawing surface is frustrating [14] and inefficient in some contexts [1]. Toolglasses are currently being implemented in IsaViz as an addition to the standard palette (Figure 7). Their use is also considered as clipboard objects holding and assigning styling data for manual GSS editing of RDF graphs.

Semantic pointing [8] improves target acquisition by assigning two sizes to targets: one in visual space adapted to the amount of information it conveys, one in motor space adapted to its importance for manipulation. Its implementation is underway in ZVTM, taking advantage of the decoupling of the system's and toolkit's cursors for adapting the control-display ratio.

7. Conclusion

We have introduced ZVTM, an open source toolkit [18] aimed at building the complex interface components found in visual language environments, meta-tools and other structured graphics editors. Using two applications (a visual language environment and a domain-specific graph editor), we have shown how ZVTM components and off-the-shelf interaction techniques can be used and combined to address several common issues faced by visual language environment designers while minimizing implementation effort.

The toolkit is used in some widely distributed applications (e.g. IsaViz), and continues to mature and grow. While informal feedback from the user community leads us to believe that ZVTM does help reducing the difficulty of implementing complex graphical user interfaces, we plan to conduct a formal experiment with master level students in order to get empirical evidence that corroborates this.

Acknowledgements

ZVTM is the continuation of XVTM, developed at Xerox Research Centre Europe and inspired by earlier research on the Visual Abstract Machine [36]. Further work on the toolkit was done during the development of IsaViz at MIT by W3C. Its development is now supported by INRIA project In Situ (http://www.inria.fr/recherche/equipes/insitu.en.html).

[4] http://zvtm.sourceforge.net/zgrviewer.html
[5] http://www.graphviz.org

References

[1] C. Appert, M. Beaudouin-Lafon, and W. Mackay. Context matters: Evaluating interaction techniques with the CIS model. In *Proceedings of HCI 2004*, People and Computers XVIII, pages 279–295. Springer-Verlag, 2004.

[2] M. Beaudouin-Lafon. Designing interaction, not interfaces. In *Proceedings of the working conference on Advanced visual interfaces*, pages 15–22, 2004.

[3] M. Beaudouin-Lafon and H. M. Lassen. The architecture and implementation of CPN2000, a post-wimp graphical application. In *UIST'00*, pages 181–190, 2000.

[4] B. B. Bederson, J. Grosjean, and J. Meyer. Toolkit design for interactive structured graphics. *IEEE Transactions on Software Engineering*, 30(8):535–546, 2004.

[5] J. Bertin. *Semiology of Graphics*. Number 0299090604 in ISBN. University of Wisconsin Press, 1983.

[6] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses:the see-through interface. In *20th annual conference on Computer graphics and interactive techniques*, pages 73–80, 1993.

[7] A. Blackwell. *Your Wish is My Command: Giving Users the Power to Instruct their Software*, chapter SWYN: a visual representation for regular expressions, pages 245–270. M. Kaufmann, 2000.

[8] R. Blanch, Y. Guiard, and M. Beaudouin-Lafon. Semantic pointing: Improving target acquisition with control-display ratio adaptation. In *ACM Conference on Human Factors in Computing Systems*, pages 519–526, 2003.

[9] M. M. Burnett, M. J. Baker, C. Bohus, P. Carlson, S. Yang, and P. van Zee. Scaling up visual languages. *IEEE Computer archive*, 28(3):45–54, 1995.

[10] W. Buxton and B. Myers. A study in two-handed input. In *SIGCHI conference on Human factors in computing systems*, pages 321–326, 1986.

[11] P. Carlson, M. Burnett, and J. Cadiz. A seamless integration of algorithm animation into a declarative visual programming language. In *Advanced Visual Interfaces*, 1996.

[12] M. S. T. Carpendale and C. Montagnese. A framework for unifying presentation space. In *UIST'01 Conference Proceedings*, pages 61–70, 2001.

[13] B. Chang and D. Ungar. Animation: from cartoons to the user interface. In *UIST'93*, pages 45–55, 1993.

[14] W. Citrin. Requirements for graphical front ends for visual languages. In E. P. Glinert and K. A. Olsen, editors, *Proc. IEEE Symp. Visual Languages*, pages 142–150, 1993.

[15] D. Cox, J. Chugh, C. Gutwin, and S. Greenberg. The usability of transparent overview layers. In *CHI'98, ACM Conference on Human Factors in Computing Systems*, 1998.

[16] S. A. Douglas and A. E. Kirkpatrick. Model and representation: The effect of visual feedback on human performance in a color picker interface. *ACM Transactions on Graphics*, 18(2):96–127, 1999.

[17] IsaViz: A visual authoring tool for RDF, December 2004. http://www.w3.org/2001/11/IsaViz/

[18] ZVTM: Zoomable visual transformation machine, January 2005. http://zvtm.sourceforge.net

[19] J.-D. Fekete. The infovis toolkit. In *10th IEEE Symposium on Information Visualization*, pages 167–174, 2004.

[20] T. Green and M. Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages and Computing*, 7(2):131–174, January 1996.

[21] J. Heer, S. K. Card, and J. A. Landay. prefuse: a toolkit for interactive information visualization. In *CHI 2005, ACM Conference on Human Factors in Computing Systems*, 2005.

[22] S. E. Hudson and J. T. Stasko. Animation support in a user interface toolkit: Flexible, robust, and reusable abstractions. In *ACM Symposium on User Interface Software and Technology*, pages 57–67, 1993.

[23] T. Igarashi and K. Hinckley. Speed-dependent automatic zooming for browsing large documents. In *UIST*, pages 139–148, 2000.

[24] H. Lieberman. Powers of ten thousand: navigating in large information spaces. In *7th annual ACM symposium on User interface software and technology*, November 1994.

[25] N. Liu, J. Hosking, and J. Grundy. Integrating a zoomable user interfaces concept into a visual language meta-tool environment. *IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 38–40, September 2004.

[26] B. A. Myers, R. G. McDaniel, R. C. Miller, A. S. Ferrency, A. Faulring, B. D. Kyle, A. Mickish, A. Klimovitski, and P. Doane. The amulet environment: New models for effective user interface software development. *Software Engineering*, 23(6):347–365, 1997.

[27] Z. Nianping, J. Grundy, and J. Hosking. Pounamu: a meta-tool for multi-view visual language environment construction. *IEEE Symposium on Visual Languages and Human-Centric Computing*, September 2004.

[28] K. Perlin and D. Fox. Pad: An alternative approach to the computer interface. *Computer Graphics*, 27:57–64, 1993.

[29] E. Pietriga. Styling rdf graphs with GSS. *XML.com*, December 2003. http://www.xml.com/pub/a/2003/12/03/gss.html.

[30] E. Pietriga, V. Quint, and J.-Y. Vion-Dury. VXT: A visual approach to XML transformations. In *ACM Symposium on Document Engineering*, pages 1–10, November 2001.

[31] S. Pook, E. Lecolinet, G. Vaysseix, and E. Barillot. Context and interaction in zoomable user interfaces. In *Advanced Visual Interfaces*, pages 227–231, 2000.

[32] G. G. Robertson, S. K.Card, and J. D.Mackinlay. Information visualization using 3D interactive animation. *Communication of the ACM*, 36(4):56–71, 1993.

[33] P. J. Rodgers and N. Vidal. Graph algorithm animation with Grrr. In *Agtive99: Applications of Graph Transformations with Industrial Relevance*. Springer-Verlag, 2000.

[34] J. Stasko. The path transition paradigm: A practical methodology for adding animation to program interfaces. *Journal of Visual Languages and Computing*, 1(3):213–236, 1990.

[35] E. R. Tufte. *Visual Explanations*. Number 0-9613921-2-6 in ISBN. Graphics Press, 1997.

[36] J.-Y. Vion-Dury and F. Pacull. A structured interactive workspace for a visual configuration language. In *Proceedings of Visual Languages 1997*, pages 132–139, 1997.

[37] J.-Y. Vion-Dury and E. Pietriga. A formal study of a visual language for the visualization of document type definition. In *IEEE Symposia on Human-Centric Computing (HCC)*, pages 52–59, September 2001.

[38] W3C. RDF (Resource Description Framework), October 2004. http://www.w3.org/RDF/.