VKB 3 Data Layer: Enhancing Visualization and Scalability

Haowei Hsieh and Frank Shipman
Center for the Study of Digital Libraries & Department of Computer Science
Texas A&M University
College Station, TX 77843-3112, USA
+1-979-862-3741
{haowei, shipman}@cs.tamu.edu

ABSTRACT
Years of extending the Visual Knowledge Builder (VKB) for new research purposes has identified limitations in how VKB represents and stores information and its visual characteristics. The VKB 3 framework adds a level of abstraction between visual data and its presentation to improve VKB’s visualization capabilities and includes an inline database in order to support the management of larger information collections.

Categories and Subject Descriptors
H.5.4 [Information Interfaces and Presentation]: Hypertext / Hypermedia

General Terms
Design, Human Factors.

Keywords
Spatial Hypertext, System Framework.

1. INTRODUCTION
The Visual Knowledge Builder (VKB) [3] is a spatial hypertext system that implements information workspaces where data can be represented as visual information objects. Information content inside VKB is stored in Information Objects and presented as Visual Symbols. Over the past few years, VKB was constantly updated with new features to support a variety of hypertext research. We have encountered a few limitations with the current data model and have since worked on revising the internal framework necessary for desired expansions to VKB. Two weaknesses of VKB 2 were its inability to support managing collections with many thousands of information objects and its inability to dynamically vary the visualizations of information objects. To address the problem of scale, we have extended the VKB framework to include a low-level data abstraction layer enabling the integration of an optimized data storage system as a backend. To address the lack of visualization options we added an intermediary layer, composed of a set of renderers and views, separating the visual presentation of VKB symbols from the storage of their visual characteristics. This paper describes the limitations and the revised VKB framework.

2. ISSUES WITH OLD FRAMEWORK
In earlier versions of VKB, an abstract data hierarchy was used to represent the separation of visual symbols and their presentation. Even so, most data was referenced by the top of the data hierarchy as this meant that access to all layers of data structures associated with a visual symbol was easy and efficient.

As a result, visual symbols served three purposes in VKB 2: first, they presented the information; second, they contained and stored visual properties of the presentation; and third, they provided user interfaces to access the information objects’ content. Essentially, the symbol is the user interface for the information object. Figure 1 shows the framework of VKB 2.

As described, visual symbols in VKB 2 included a combination of storage, display, and interface functions even though they were abstracted away from the text, metadata, and other content of information objects. In VKB 2, symbols acted as both the view and controller for the information object model. Additionally, any visual features that were assigned to the symbol that did not affect underlying information object content was also stored by the symbol. This led to several problems:

1. Visual symbols became heavy weight objects and relatively expensive for frequent access. As is generally true for interactive visualizations, redrawing and refreshing of the VKB workspace is frequent. This meant that system performance was notably affected when there were hundreds of visual symbols.

2. In situations where a different view of a symbol is required (e.g., different scale) the symbol’s visual characteristics had to be modified to reflect the change in its view.

3. The tight bond between visual symbols and the layout and other visual information also generated overhead when many visual updates in the space were necessary. Because the visual symbol is the presentation and interface widget, updating visual symbols individually could result in many redraw events unless optimized by the developer.
An example of this third problem occurs with the VKB history mechanism [2] which allows users to navigate to any point of time in the space. VKB2 accomplishes history navigation by stepping through each edit (called a history event) of the space to ensure consistency. This results in many updates to visual symbols and their associated redraws unless we build our own drawing subroutine. We mitigated this problem in VKB 2 by turning off refreshes in the interface while stepping through edits but this still meant that the heavy weight widgets were being updated repeatedly, even though it was not visible to the user.

Besides the above issues of scale, another issue with the integration of the model, view, and controller aspects of a visual symbol was that many aspects of the visualization of an information object were pre-determined by the symbol itself (position, size) and only a few content visualizations (full text, image, and thumbnail) were supported. This design limits the ability to add alternative visualizations for objects and makes it very cumbersome to add alternative layouts for collections of information objects.

3. NEW PLAN

VKB 3 is being designed to address the above problems by more truly separating the presentation, storage, and interfaces for information, even when that information is visual in nature (e.g. the position, color, etc. of a visual symbol). VKB 3 includes a dedicated data storage layer to hold the information and adds a view abstraction that separates the presentation of and interaction with visual symbols from the data storage layer. Views are associated with one or more rendering engines that determine what aspects of the stored information are used in the visualization and how. Figure 2 depicts the new framework.

For the data layer, an in-process database is embedded within VKB so users do not need to know about or install the database. The database engine is configured to hold all workspace data in memory for performance at the current time but later can be configured to only maintain part of the information in memory and rely on the file system if memory becomes a problem.

In addition to the data storage layer, a data cache layer is added for three purposes: firstly, it is used to hold frequently accessed information to reduce the overhead that comes with database access. Secondly, it is designed to isolate the database subroutines from the rest of the application components. Thirdly, each data cache can completely or partially duplicate the content of the database and be assigned with individual viewing profiles to render each sub collections.

This framework allows for flexible control over the view for workspaces and sub-spaces (collections). For example, a workspace can have several pre-defined views for the information it contains, like the existing textual and thumbnail views, and users can switch views easily. In addition, each collection can also have its own views, leaving the freedom of defining algorithmic methods for assigning visual properties (position, color, etc.) based on the information object content. The result is the ability to add a more powerful two-way mapping engine as was used in VITE [1] in place of the rendering engine.

The greater separation of visual workspace and the storage layer also makes it possible to complete large-scale updates before any updates to heavy weight interface components are instantiated. One good use of this is the previously mentioned history traversal problem. Because batch transactions can be applied to the workspace information inside data cache, the workspace does not need to refresh through intermediate states during the update. The final workspace will be rendered by the rendering engine only when the history list is completely updated into the data cache. Another use of this ability is to maintain multiple versions of the workspace at one time. By duplicating a workspace’s data cache, multiple versions of a spatial hypertext could be opened and manipulated simultaneously.

4. SUMMARY

VKB’s symbol-object representation was originally meant to separate content from its presentation. In visual workspaces like VKB, visual characteristics encode significant information in addition to the original content that must be stored and manipulated much like the original information content. The VKB 3 framework is designed to more efficiently support transient visual changes while keeping essential visual information intact. This new version is still in its early stage of implementation, and we anticipate performance penalties and other problems due to the added levels of abstraction. However, we expect the new framework to scale better as information space sizes and to provide a more flexible platform for future spatial hypertext research. Look for VKB 3 at http://www.csdl.tamu.edu/VKB.

5. REFERENCES

