A Declarative Approach for Information Visualization

Adriane Kaori Oshiro
Andrea Rodrigues de Andrade
Maria da Graça Pimentel

Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo
Av. Trabalhador São-Carlense, 400 – Caixa Postal 668
13560-970, São Carlos, SP

{kaori, aandrade, mgp}@icmc.usp.br

Abstract. *Information Visualization investigates the use of visual and interactive information representations with the aim of reducing users' cognitive overhead as they analyze information. The objective of this work is to investigate mechanisms that allow presenting a large amount of information in Web-based platforms. We have built the iVIEW infrastructure that: (a) defines a declarative language based on XML Schema specifying a SVG-based visualization layout for information contained in XML documents; and (b) describes algorithms that execute the necessary steps to obtain a graphic representation of the information, implemented using XSLT. In this paper we describe the information elements and the visualization structures of iVIEW along with user-interaction resources. We also show the necessary steps for obtaining a graphic representation of the information using SVG.*

Keywords: Information visualization, XML-based language definition and processing, SVG.

1. Introduction

The recent growth of the Internet as a way to obtain information in the context of several application domains has demanded the incorporation of visual techniques that aid users in the task of interacting with this vast universe of information in an efficient and intuitive manner. Information Visualization has emerged as a research area that investigates the use of visual and interactive information representations with the aim of reducing users' cognitive overhead as they analyze information [Card et al., 1999].

The Web has become a repository for publishing applications such as newspapers and magazines, project-related documentation and educational material in general. One important feature in such publishing applications is that all information ever published is usually made available for users. As a result, the amount of information that a user has available to review grows as the time passes. In the case of a newspaper, for instance,

a user may be allowed to review an issue as old as needed. This is a typical application for visualization tools: to allow users to access and have some understanding of a huge amount of information. Moreover, it is important to offer users with not only a way to view all the information but also filter how the information is to be presented according to some specific attributes.

As far a typical publishing application is concerned, it is important to consider that each issue of a newspaper, for instance, is intrinsically related to others that have been previously published: this reflects the constant evolution of the contents of the published material. In such contexts, it is important that the visualization of the information of all issues be able to express the intrinsic relationship existent among the separate issues. Most visualization tools allow users to visualize existing relationships among the information items(e.g. [Hibino and Rundensteiner, 1996]), but they are associated to data with a specific structure and particular to specific domains [Polys, 2003].

SVG (Scalable Vector Graphics) is an XML based markup language for the specification of vector graphics, such as circles and polygons [W3C, 2003]. SVG documents can be visualized as a *standalone* document or embedded in HTML documents through the use of browser *plugins*. Several applications that use SVG can be found in the Web. For instance, Southard proposes the use of SVG to represent the structure of HTML and XML documents through an interactive SVG tree: branches correspond to document elements and the leaves correspond to attributes of these elements; by positioning the mouse cursor over a branch or a leaf, the name of an element or the content of its attribute is presented [Southard, 2001]. Examples by Adobe include graphs and images that simulate 3-D representations of molecules and interactive presentations of buildings and theaters [Adobe, 2002].

Applications that use SVG can present data from external sources, such as XML documents or Relational Databases [North et al., 2002]. For example, XML documents can be processed by algorithms contained in XSLT document in order to extract pertinent information towards generating SVG documents. This process was exploited in the implementation of a flexible and domain-neutral infrastructure, called iVIEW.

The main goal of this work is the investigation of mechanisms that exploit the processing of structured documents to allow the visualization of a large amount of information on the Web at the same time that supports a degree of interaction for presenting intrinsic relationships. We present iVIEW, that provides a mechanism for the visualization of information by means of automatically processing XML-based structured documents towards generating interactive SVG representations. Supported by a declarative language specified in XML Schema, iVIEW is extensible and independent of application domain by the use of XML format for data input.

In Section 2. we describe the iVIEW infrastructure while Section 3. details its use in the context of an XML publishing framework. Section 4. discuss our approach in the context of related work. Section 5. presents our conclusions and future work.

2. The iView infrastructure

The main goal of the iVIEW infrastructure is to provide the visualization of information extracted from XML documents using graphic representations in SVG. In order to achieve that goal, we have defined: a) a declarative language specifying a visualization layout for information (*layout.xml* in Figure 1); and b) algorithms to execute the necessary processing steps of documents to obtain a graphic representation of the information.

Processing steps. The processing of documents in iVIEW occurs in two stages. The first stage allows an intermediary transformation format to be obtained for a specific application (*Step 1a* in Figure 1). In the second stage, the intermediary specification is processed towards generating a final presentation specification (*Step 2* in Figure 1). It is this two-stage processing that gives generality to the overall transformation. Because the target documents of the overall processing are SVG, for graphics, and JavaScript, for user-interaction, the first stage also generates an interaction document (*Step 1b* in Figure 1).

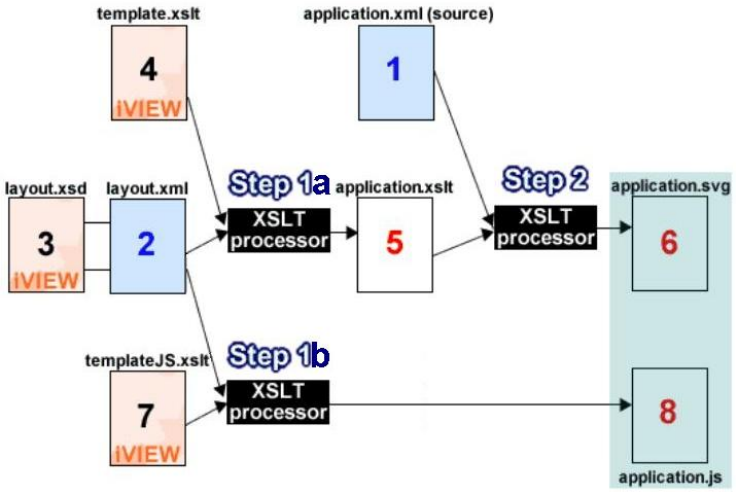

Figure 1: Documents, resources and processing steps of iVIEW for a graphic representation of information in SVG.

The numbered rectangles in Figure 1 represent input and output documents that are processed during each step. The developer is responsible for creating instances of *application.xml* (1), which correspond to the source of information that is to be visualized. The developer also designs *layout.xml* (2) that defines the structure of the presentation of the information, according to *layout.xsd* (3). Given these both input documents, the developer receives: *application.svg* (6) that contains a graphic representation in SVG of *application.xml* (1); and *application.js* (8) that contains JavaScript functions for providing users' interaction with *application.svg* (6).

The *application.svg* (6) and *application.js* (8) documents are generated by means of the iVIEW two-stage processing. In *Step 1a*, the generic XSLT stylesheet *template.xslt* (4) generates the XSLT stylesheet *application.xslt* (5) according to the specifications con-

tained in *layout.xml* (2). In order to generate *application.js* (8), *layout.xml* (2) is processed by means of the fixed XSLT stylesheet *templateJS.xslt* (7) in *Step 1b*. The *application.xslt* (5) stylesheet is specific for *application.xml* (1) to generate *application.svg* (6) in *Step 2*.

Visualization structure and groups of elements of iVIEW. In order to provide the visualization of information extracted from XML documents we have divided the basic visualization structure of iVIEW into two main areas: a) an area to present visualization elements groups with their attributes and relationships; and b) an area containing selection filters to show or hide determined elements. The actual positioning and dimensions of those areas are defined by the developer. Figure 2 are two examples of how the iVIEW visualization structure can be configured.

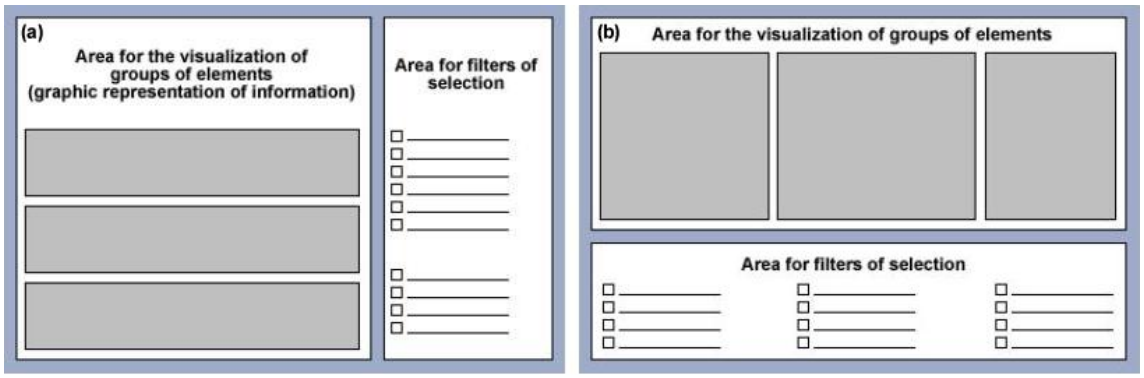

Figure 2: Example of iVIEW basic visualization structure configuration. a) Vertical display of main areas. b) Horizontal display of main areas.

We have defined a group of elements of visualization as a set of XML elements with same characteristics and structure, same attributes and that execute the same actions as response to users' interaction. For instance, the teaching staff of a given university can be considered as a group, where each instructor is associated to a visualization element. Assuming that all elements have the attributes "name", "department" and "email", we can say that they have the same nature and same characteristics. Figure 3 presents a possible visualization of three distinct elements groups:

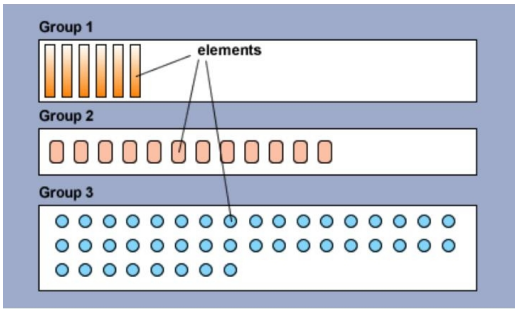

Figure 3: Distinct groups of elements.

As we can see in Figure 3, the elements are represented graphically by SVG shapes, such as bars, rectangles and circles. Each element represents an item of information, which contains one or more attributes that can be related to other elements. An important feature of groups of elements is the fact that all elements can be related to each other, in comparison to specific characteristics. This relationship can be visualized through filters and interaction functions.

Eventually, the amount of information items presented to users can be large and there are cases in which users may be interested in visualizing only elements with attributes containing a determined value. In order to provide the option of visualizing only determined elements, we have implemented elements selection filters. In this case, elements common attributes allow the visualization of intrinsic references among them.

We can also consider the visualization elements groups as structured lists. Therefore, the use and the combination of selection filters over these element groups can simulate the effect of union or intersection of lists. In Figure 4 we can observe an example of a selection filter. In Figure 4.a we can see a students group and a filter that select these students according to their graduation course. In Figure 4.b the filter is being used, and only students from ”Computer Science” and ”Physics” courses are visualized.

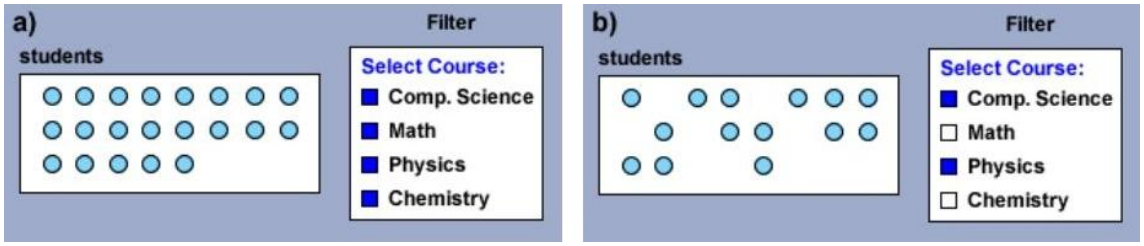

Figure 4: Elements selection filter. a) All students from all courses are shown. b) Only students from ”Comp. Science” and ”Physics” are shown.

Layout language. The goal of the first step for obtaining a graphic and interactive representation of an information set is the generation of a specific XSLT stylesheet (*application.xslt*) containing instructions that will generate a SVG representation of the associated information. This specific XSLT document will parse data to be visualized from the document *application.xml*.

An XSLT processor is used to generate the document *application.xslt*. As input, the developer must create the *layout.xml* document, based on the layout language defined in *layout.xsd*, that contains: (a) a specification of the general layout to be shown; (b) the layout of elements of information that will be visualized; and (c) some parameters necessary to establish the relationship between the elements.

Basically, the document *layout.xsd* defines the three main parts that establish the aspects and the properties of the final visualization: a) the layout, that makes possible the configuration of dimensions, positioning and colors of the main window, as well as

the SVG shapes that will represent information elements; b) the elements, which contains information of how the visualization elements are shown and what interaction functions can act over them; c) the filters, which contains information about the appearance and behavior of filters that are used.

As an example, the following code is a portion of a document *layout.xml* that refers to data about captured sessions in classrooms, references, students and instructors of a determined university:

```
<layout>
  <window  width="770" height="470" bgcolor="#648EB0"/>
  <title label="1o.Season 2001 - 2o.Season 2002"
    font_face="verdana" font_size="16" font_color="black"/>
  <script file="courses.js"/>
  <bars>
    <bar name="captured_sessions" stroke="3" height="30"/>
    <bar name="references" stroke="4" height="32"/>
  </bars>
  <circles>
    <circle name="instructors" radius="4" stroke_width="1"/>
    <circle name="students" radius="4" stroke_width="1"/>
  </circles>
</layout>
```

In the example, the attributes of the *window* element define the width and height of the final visualization area, as well as its background color. The *script* element contains the attribute file that defines which JavaScript document will be used to provide user interaction with the final visualization (the JavaScript document *application.js* is automatically generated). Finally, the *bar* and *circle* elements define how the visualization elements will be represented: captured sessions and references will be represented by bars; students and instructors will be represented by circles. Eventually, this representation could use any other basic shape of SVG, such as rectangles or other polygons.

In another part of *layout.xml*, the elements portion establishes the layout and elements properties that will be visualized:

```
<elements>
  <element path="courses/references" groupid="refs">
    <area x="10" y="320" width="545" height="40"
      stroke_width="2" stroke_color="white" bgcolor="#C6D5E1"/>
    <title name="references" x="10" y="315" font_color="white"
      font_size="11"/>
    <representation name="references" stroke_color="black"
      color="white"/>
    <initial_position min="1" max="50" x="18" y="324"/>
    <shift>10</shift>
    <attributes>
      <attribute name="title"/>
      <attribute name="author"/>
      <attribute name="course"/>
    </attributes>
  </element>
</elements>
```

The *element* element refers to information elements groups contained in the XML document *application.xml* — for instance, captured sessions, references, instructors and

students — and contains the definition of how these elements will be presented. The attribute path shows the hierarchical location of elements within the XML document *application.xml* that, in this example, are the elements containing data about references.

The attribute *groupid* contains an identifier to this group of elements and will be used subsequently by interaction functions with users. The *area* element defines the dimensions and properties of the space occupied by these elements in the final visualization. The *representation* element associates elements to their SVG representation defined previously in the general layout part (in the case of references, it was defined that they would be represented by a bar named ”references”). The *initial_position* and *shift* elements indicates, respectively, where the first element must appear and the space in pixels between each element. Finally, the attributes that will be visualized are defined by the *attributes* element.

The following code is the third and last part of the *layout.xml* document that is being presented as an example in this section, where the properties of the filters of selection of elements are defined:

```
<filters present="yes">
  <title label="Filters" font_size="14" font_color="blue"
    x="645" y="25"/>
  <area x="565" y="10" width="195" height="450" stroke_width="2"
    stroke_color="white" bgcolor="#98B4CB"/>
  <filter name="references" descriiption="Hide/show references:"
    font_color="blue" font_size="13" x="550" y="0">
    <groupelem groupid="refs" attributeid="2"/>
    <item label="Hypermedia" color="black" size="12"/>
    <item label="Multimedia" color="black" size="12"/>
    <item label="HCI" color="black" size="12"/>
    <item label="OS" color="black" size="12"/>
  </filter>
</filters>
```

The attribute *present* of the *filters* element indicates the presence of filters in the final visualization, since they may not be present if the developer defines so. The *title* and *area* elements define, respectively, the title and the area occupied by selection filters. Each filter used in the developer application is defined towards their properties. In this example, the filter named ”references” will act over the elements group which attribute *groupid* is ”refs”. The selection is composed by item which attributes label contain the values ”Hypermedia”, ”Multimedia”, ”HCI” and ”OS”. Therefore, users can visualize only the references about the course ”Hypermedia”, or about the courses ”Multimedia”, ”HCI” and ”OS”, or any combinations they prefer.

In order to guarantee that the *layout.xml* document is valid, it is necessary to follow the definitions contained in the *layout.xsd* document. This XML Schema defines the sequence and the possible number of occurrences of each element within the correspondent XML instance, as well as the data type that each element and attribute can have.

After the *layout.xml* document is properly specified, it will be processed by an XSLT processor together with the generic XSLT document *template.xslt*. The *template.xslt* document has the function of using the specifications contained in the *layout.xml* document

to generate the *application.xslt* document which, in turn, will process the *application.xml* document. As a result, a SVG document (*application.svg*) is produced and it presents the information of the document *application.xml* in the format specified by the developer.

The portions of documents presented in this subsection refer to an application for visualization of data about captured sessions in classrooms, references, students and instructors of a university. The XML document for this application (*application.xml*) is relatively large and information in it cannot be visualized in a single screen. However, the visualization of the information in a single screen can be possible by means of the transformation of the XML document into SVG.

Figure 5 is the SVG representation of the application in its complete form, where the information elements are represented by vertical bars (captured sessions and references) and circles (instructors and students), according to parameters defined by the developer in the first step. Figure 5 also shows the execution of generated interaction functions obtained in the last step.

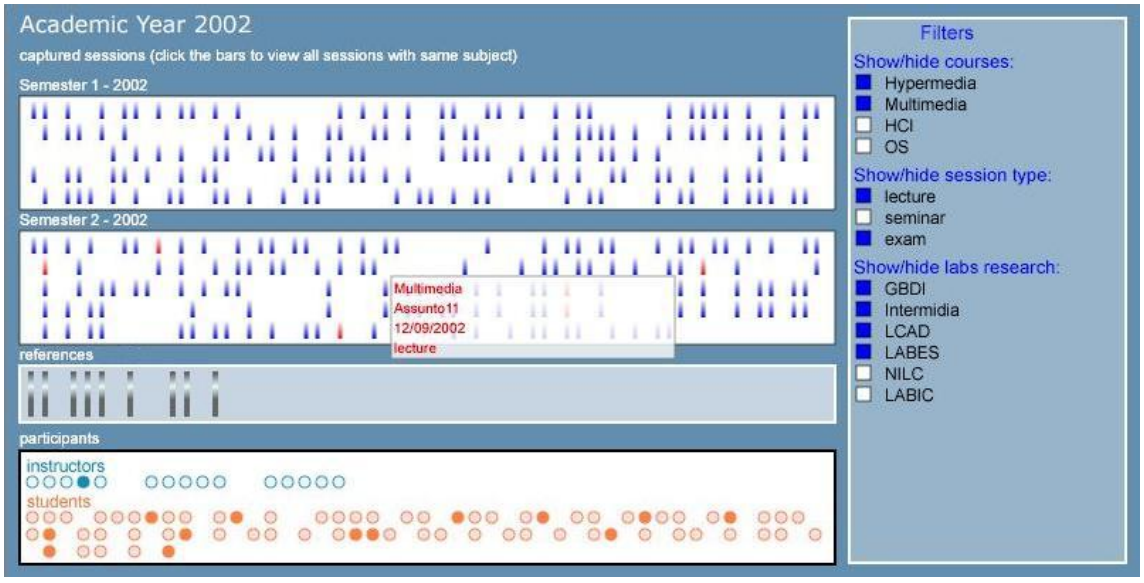

Figure 5: SVG representation associated to a JavaScript document with functions that provide interaction with the elements displayed.

3. iVIEW in use

As the iVIEW infrastructure is XML-based, its processing steps could be implemented in an XML publishing framework. We exploit the Cocoon Java Framework, developed by the Apache Software Foundation, that supports Web publishing based on the processing of XML documents [Apache, 2002].

The processing in Cocoon is pipeline-based: as a result of a user request for a document, XML documents enters the pipeline, are processed and passed by means of

SAX events to the next processor in the pipeline, until they exit the pipeline in a format that can be delivered over the Web. The following is a pipeline defined according to iVIEW processing steps[1]:

```
<map:pipeline>
  <!-- Step 1a -->
  <map:match pattern="application.xslt">
    <map:generate src="layout.xml"/>
    <map:transform src="template.xslt"/>
    <map:serialize type="xml"/>
  </map:match>
  <!-- Step 1b -->
  <map:match pattern="application.js">
    <map:generate src="layout.xml"/>
    <map:transform src="templateJS.xslt"/>
    <map:serialize type="text"/>
  </map:match>
  <!-- Step 2 -->
  <map:match pattern="application.svg">
    <map:generate src="application.xml"/>
    <map:transform src="cocoon:/application.xslt"/>
    <map:serialize type="svg"/>
  </map:match>
</map:pipeline>
```

4. Related work

Several work in the context of information visualization exploit the use of XML based languages to store and transform data. One example is an application that uses XML and XSLT to manipulate video information [Christel et al., 2001]. This application used Java applets to display information obtained as result of queries. Our implementation is based on the specification of XML documents that are processed by general XML processors.

Other example is an application that provides the visualization of documents that embed contextual, data-driven information components using SVG [Weber et al., 2002]. However, the document generation is manual or semi-automatic; one of the main efforts of our implementation was to provide automatic generation of SVG representations and other documents related user-interaction.

XML-based languages are also exploited by an application that uses X3D and VRML to visualize data stored in CML (Chemical Markup Language), which is an XML-based language for representing chemical data [Polys, 2003]. XSLT is used to transform information into X3D and VRML. In the same context, data stored in CML has also been represented by interactive graphics in SVG [Adobe, 2002]. Our implementation is more generic in terms of information domain, since any XML specification can be used. Moreover, our work supports the use of any SVG-based graphic representation.

[1]Example available from http://iclass.icmc.usp.br/iview

5. Conclusions and further work

To Web users, the task of finding, interpreting and interacting with a vast universe of information is a non-trivial task. Information Visualization is a research area with the objective to provide interactive ways to represent data.

Supported by a declarative language specified in XML Schema, the iVIEW infrastructure allows the developers to define graphic SVG information representations so that users can view a great amount of elements in a single visualization in an interactive way.

Compared to current efforts, our implementation is (a) more general in terms of the format of the input data specified in XML documents; (b) does not require the developer to specify XSLT transformations, JavaScript functions or SVG representations; (c) allows reuse of presentation templates as layout documents; (d) allows the identification of relationships among items of information towards the visualization of the overall information.

Possibilities to extend the resources offered by the current infrastructure include: to investigate ways to allow the establishment of information relationship after the generation of the SVG representation; to investigate other resources that SVG is capable to provide so we can add new interaction functions; to create graphic publishers to define the visualization layout.

References

Adobe (2002). SGV Zone Demos. http://www.adobe.com/svg/demos/main.html.

Apache (2002). The Apache Software Foundation: Apache Cocoon 2.0. URL:http://cocoon.apache.org/2.0/.

Card, S. K., Mackinlay, J. D., and Shneiderman, B. (1999). *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers.

Christel, M. G., Maher, B., and Begun, A. (2001). XSLT for tailored access to a digital video library. In *1st. ACM/IEEE-CS joint conference on Digital libraries*, pages 290–299.

Hibino, S. and Rundensteiner, E. A. (1996). MMVIS: A MultiMedia Visual Information Seeking Environment for Video Analysis. In *Proc. ACM Multimedia'96 Conference*, pages 15–16.

North, C., Conklin, N., and Saini, V. (2002). Visualization Schemas for Flexible Information Visualization. In *Proc. IEEE InfoVis 2002 Symposium*, pages 15–22.

Polys, N. F. (2003). Stylesheet transformations for interactive visualization: towards a Web3D chemistry curricula. In *Proc. 8th. International Conf. on 3D web technology*, pages 85–90.

Southard, J. (2001). XML Grove. http://www.jeffsouthard.com/demos/grove/.

W3C (2003). World Wide Web Consortium, Scalable Vector Graphics (SVG) 1.1 Recommendation. http://www.w3.org/TR/SVG11.

Weber, A., Kienle, H. M., and Müller, H. A. (2002). Live documents with contextual, data-driven information components. In *Proc. 20th annual international conference on Computer documentation*, pages 236–247.