Using SVG for an User-Interface to a Digital Library

Harald Krottmaier, Christian Scherngell

Institute for Computer Graphics and Knowledge Visualization, Graz University of Technology
Infeldgasse 16c, 8010 Graz, Austria
e-mail: h.krottmaier@cgv.tugraz.at; grave@sbox.tugraz.at

Abstract

Interfaces for users on the web are usually created using standard HTML-technologies. Although dynamic text-based features are widely available on the web (e.g. by using DHTML) there are still some limitations in the user experience when using dynamic graphical elements. In this article we focus on a GUI-prototype using Scalable Vector Graphics (SVG) as an interface to an existing Digital Library. First we describe the current situation of the running system and explain some of the limitations in the GUI. Thereafter we compare the document-format with other electronic formats. An overview of the prototype and a description of the used framework will be given in the main section of this article. Finally we are going to show an improved version of a search task using an SVG-category viewer. The currently supported SVG-recommendation (SVG 1.1) by the SVG-plugins have some limitations. The upcoming release will solve some of the problems and introduce new features related to data-handling and database access. An outlook of these new features and lessons learned in implementing the prototype will close the article.

Keywords: SVG; user-interface design; GUI-framework; web-based interface

1 Introduction

In this article we are going to report on a GUI-prototype, which was implemented in context of a master thesis using SVG technology. The Digital Library that was used as backend to this application is now running for more than 11 years and is based on a Hyperwave Information Server (HWIS). HWIS is a knowledge management-system using an object-oriented database layer. Every entity that is stored in the database has metadata attached to it. Other features such as similarity match, indexing of different electronic content formats, and the possibility to group objects in collections (beside many more features which are available out of the box) make this platform an ideal basic technology for the development of Digital Libraries.

In the Journal of Universal Computer Science (J.UCS), most navigational features are created on the fly using data from the database. It's easy (i.e. a simple API-call) to extract information about relations of objects such as parent-child relations.

We serve different types of objects to the user: volumes, issues, and papers. Volumes are collections (i.e. containers) for issues. Issues are containers for papers, and papers are containers for the content itself. Different electronic formats are stored on the server and therefore users may select the most appropriate one. Currently we offer PDF- and PostScript-versions of the content and mostly also HTML-coded versions of the papers. Papers are also categorized using ACM-Classifications. A more detailed description of the architecture and features of this Digital Library is given in [1].

However, since HTML is used as GUI-technology to the database there are limitations in the flexibility of GUI elements. To give an example, let us take a closer look to the category-visualization: Currently categories are visualized in a flat-view, the user enters a new level, and the previous view disappears. Users are disoriented (although navigation back to the original view is provided) and are likely to be lost-in-hyperspace.

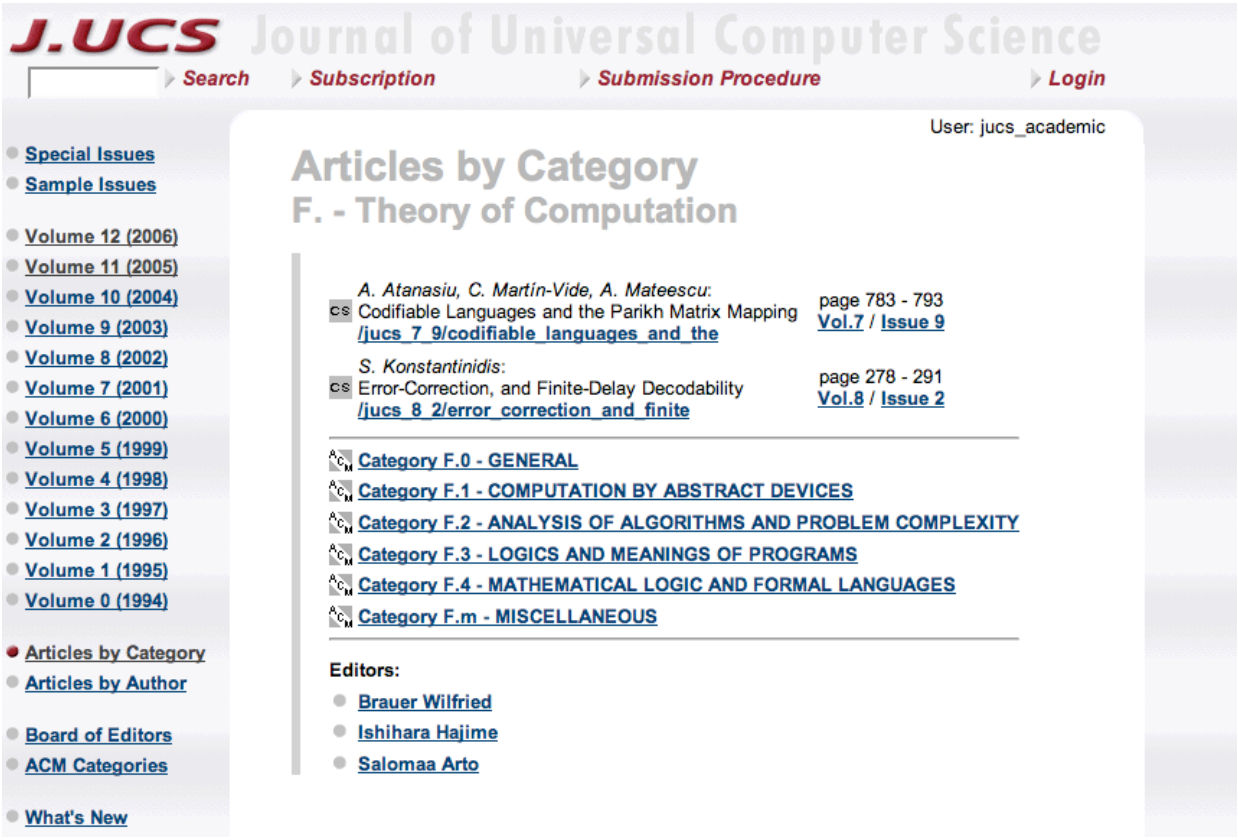

Figure 1: Typical "Flat"-view of the Papers by Category Collection

A tree implemented in HTML on the left side of the page may help users to discover and traverse the structure. However, using static images of the same size as representatives of the leaves of the tree will not help users much. They want to see immediately how many papers are in a specific category. One may argue that adding a decimal number to a leave of the tree is sufficient, however it is not easy to compare very fast numbers of papers in category A and category B. Using color, size and additionally some text is more effective.

Manipulating images on the server-side is easily done using scripted-based technology. Updates must be recalculated and reprocessed, this may be an CPU intensive task. Therefore we decided to let the images representing the leaves of the tree render on the client-side using Scalable Vector Graphics (SVG). This will reduce load on the server because all rendering tasks are performed on the client.

In the following sections we describe SVG and we give an overview of advantages of that technology. Thereafter we explain the concept and architecture of the prototype and show some screenshots of the implementation. Finally we discuss the results.

2 About SVG

The Scalable Vector Graphics (SVG) format is recommendation of the World Wide Web Consortium (W3C) for describing scalable, two-dimensional vector graphics on the Web. Simple elements are easy to describe. SVG is based on XML (Extensible Markup Language). Beside vector-graphic elements it is possible to include text, raster graphics (such as jpg- or png-images) and other multimedia-components. SVG is - as all XML-based languages - completely described in an ASCII-file containing all markups. Therefore arguments in favor of XML also count for SVG. No development environment is necessary to develop SVG-elements: a simple text-editor will do the job as well.
Additionally to static elements in SVG one may find other interesting features such as animations and interactive elements. Animations in graphics are usually implemented using GIF. Interactive (graphic) elements are a mostly implemented using static graphics and JavaScript or simple HTML-linking. In SVG the whole Document Object Model (DOM) is integrated and available to the developer. Therefore interactive modifications of the graphic can be implemented easily.

Since SVG is coded in XML every XML tool can be used to parse and/or modify the graphical element. Developers are not forced to learn new tools just to work with graphics. SVG fits perfectly in a modern XML-based environment. To render SVGs or to interpret SVG-code it is necessary to install an appropriate plug-in on the client side (i.e. web-browser).
Due to comprehensive possibilities and many advantages over comparable online and graphic formats SVG could serve in the future as basis for many graphic applications in the Web.

3 Realisation of SVG-based GUI's

SVG may be used for simple graphical elements. But other applications are also possible. Since JavaScript and DOM are available to the developer innovative web-based applications are possible. Let us now comment on some attributes of SVG.

Scalability: Graphical elements should use the whole dedicated space. If other e.g. screen-resolutions or window-sizes are used, than the layout should also automatically change and adapt to users' needs. Using raster graphic elements in different sizes hold dangers of always being in the wrong resolution. On the one hand if the output device is very large, it is likely that the image-resolution is too low. On the other hand if the application is displayed on a mobile device with very limited screen-resolution, the prepared image on the server is probably too large and the resolution is too high. In combination with the available bandwidth on the mobile device it is clear, that the application designer or developer must prepare several different images for one and the same GUI-element. Using SVG in this case will therefore reduce workload of designers and developers.

XML: Since SVG's base is XML it is easy to integrate other XML resources into SVG. XML-technologies are omnipresent nowadays in web-applications. Most of the content (and metadata) is available in XML and it is often necessary to integrate parts of content into graphics (e.g. title of a button etc.). Using SVG makes it also in this case very easy to grab the content and integrate it.

Text is Text: When using textual elements in raster graphics, characters are transformed into pixels as well. In SVG text is not transformed. Therefore it is possible for users to simply copy-and-paste text-elements out of the graphics. For search engines it is also possible to index these graphics since text is always text. Currently most developers work with metadata on images to make it possible for search-engines to index the graphic.

Style and Script: Elements in SVG are styled using well-known technology: CSS. Developers are not forced to learn yet-another-styling-language. When using ECMA-Script it is very easy to create event-driven applications.

Open Source: The SVG recommendation is available to the public. Therefore there is no need to pay license-fees when using SVG.

All these attributes shows that SVG may be used as GUI in a web-application. We will now show how we have implemented a GUI for the Journal of Universal Computer Science.

4 The Concept for the SVG-based GUI-Prototype

When we have started the project in early 2004 the number of existing SVG applications was very limited. Since there was little experience in writing large SVG applications in the community we developed a number of concepts on how to design the application. The following two main characteristics of the developed concepts were in common:

GUI should looks like an ordinary web site: Users are used to use the web-browser. They know how to handle hyperlinks, how to browse the information etc. Therefore it was clear, that the SVG application should also look like an ordinary web-site: navigation on top and the borders of the content, hyperlinks to other parts of the information-space in the content etc. Using the known web-browser to handle SVG applications makes it necessary to install a plug-in for the browser. Several companies have implemented such plug-ins for different browsers and platforms. Since users are used to install plug-ins on their computers, it is very likely that they don't need any help from administrators to view SVG-enabled sites.

Scripts must be available: It is necessary to add scripting technologies in SVG. DOM manipulations in particular are inevitable in modern applications. Since these manipulations are easily performed in SVG using JavaScript or ECMAScript, it was decided to use these manipulations.

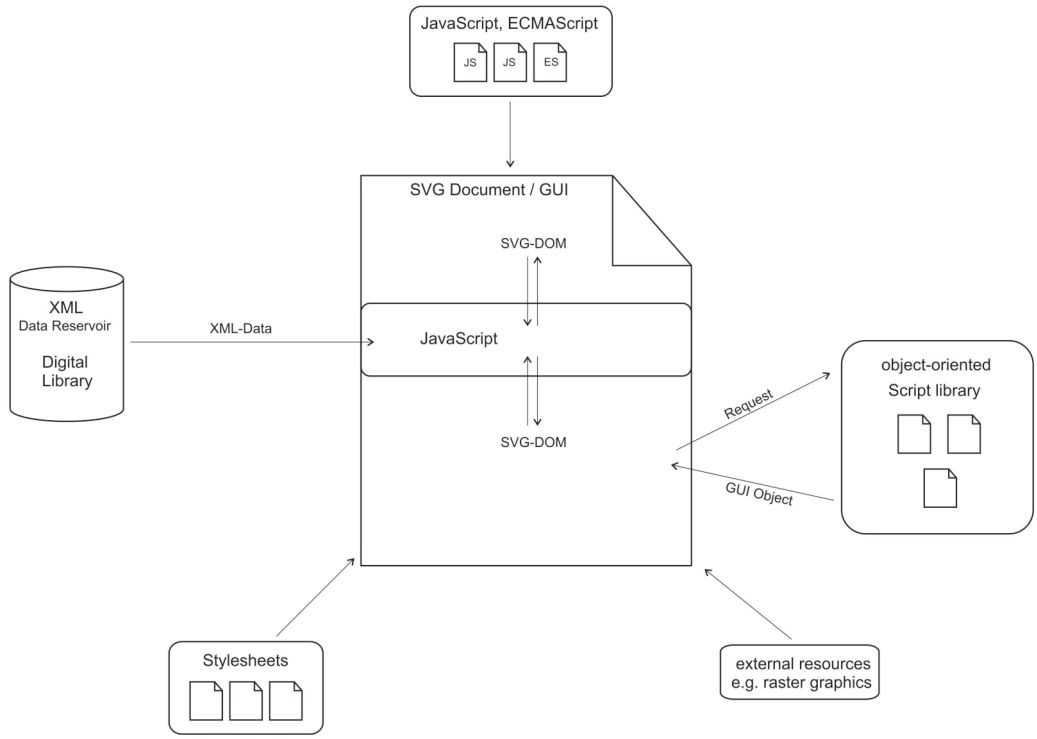

Figure 2: Basic concept for the GUI-prototype

Fig. 2 displays the basic concept: a central SVG document connects the content (which is stored in the database), the style sheets, other resources, and libraries for methods to manipulate the content (i.e. perform all DOM-manipulations).

5 Realisation of the Prototype

In this part of the article we are going to describe the implementation of the project. After we decided to use the concept displayed in the previous figure, we explored some SVG-frameworks which were available at that time. It was very clear to us that it was not possible to implement a whole framework from scratch. Since we were interested in working with SVG as a GUI we focused on GUI related tasks and not on the implementation of a framework.

The GUI Component Framework. We have searched for an open-source framework since we wanted to take a closer look not just at the functionality but also at the source-code. It was also clear that the framework must fulfill requirements related to object-oriented handling of documents. We choose the SVG Programmers Application Resource Kit (SPARK, [2], [3]) as a framework for our development. SPARK is an open-source framework that tries to set a standard for SVG-based web-applications. The core of SPARK is a flexible, extensible framework concept which helps user in the development of SVG-UIs.

Access to Content. The next SVG specification (SVG 1.2, [4]) supports developers with an interface for communication with databases. However, the current specification 1.1 (SVG 1.1, [5]) does not help developers very much in this respect. Adobe's SVG-Viewer ([6]) supports developers with a `getURL()`-implementation.

This method makes access to data using URL-addressing possible and is successfully used in the prototype. Handing of data and integration of content into the SVG application is implemented in JavaScript. The content itself is available in XML format. Therefore it is possible to get a DOM-tree representation of the content. JavaScript methods transform parts of the content to appropriate SVG-useable parts that are then integrated in the GUI.

Events (such as mouse-clicks or keyboard-actions) are handled in two different ways in the prototype: first, it is possible that an object (that receives the event) handles events triggered by the user immediately. This is the

most efficient way of handling events since that event is not handed over to another container. Usually, events that are executed by the SVG-subsystem itself are handled this way (e.g. animation-effects, color- or font-changes etc.).

Second, events can be forwarded to the SPARK-framework. So called event-handlers take care of delivering the event to the appropriate object. This type of event-handling is used when SPARK-related tasks - such as moving windows - must be performed.

As previously mentioned SVG documents may be displayed on different screen-sizes. The specification of the SVG format already implies extensive aspects concerning the scalability of documents. For example a GUI realized with SVG must be developed and optimized only for one individual set of representation parameters. SVG viewers recognize changes on the document view automatically. In further consequence this leads to an adaptation of the document content which has to be rendered to the new conditions. However, if the continuity of an SVG application additionally demands for a reaction on changes concerning the plotting area (e.g. the window size of the browser), then the application must be set in knowledge about these changes explicitly, and has to be made familiar with the desired reaction. For the prototype, managing this was realized with one in script language formulated monitoring function for web-browsers. If this function registers geometrical changes on the plotting area (the browser window), thereby necessary scaling computations become accomplished, and their results are handed over to the SVG document, where they serve then as parameters for scaling of document content that has to be visualized.

Styling of Documents. The specification of SVG allows different possibilities to render and style documents. In the developed prototype we have implemented a combination of these possibilities. Inside the SVG document we have used SVG-specific attributes for rendering which are defined in the SVG specification. To style and render single elements and to do some animations this is probably the fastest way. This type of styling allows an efficient access of elements. To style a set of elements and the overall document-view, we have decided to use external style sheets. This approach makes it easier and more efficient to change the representation of many related elements in SVG. Since Cascading Style sheets (CSS) are well known in the developer's community, we have decided to use this technology as well.

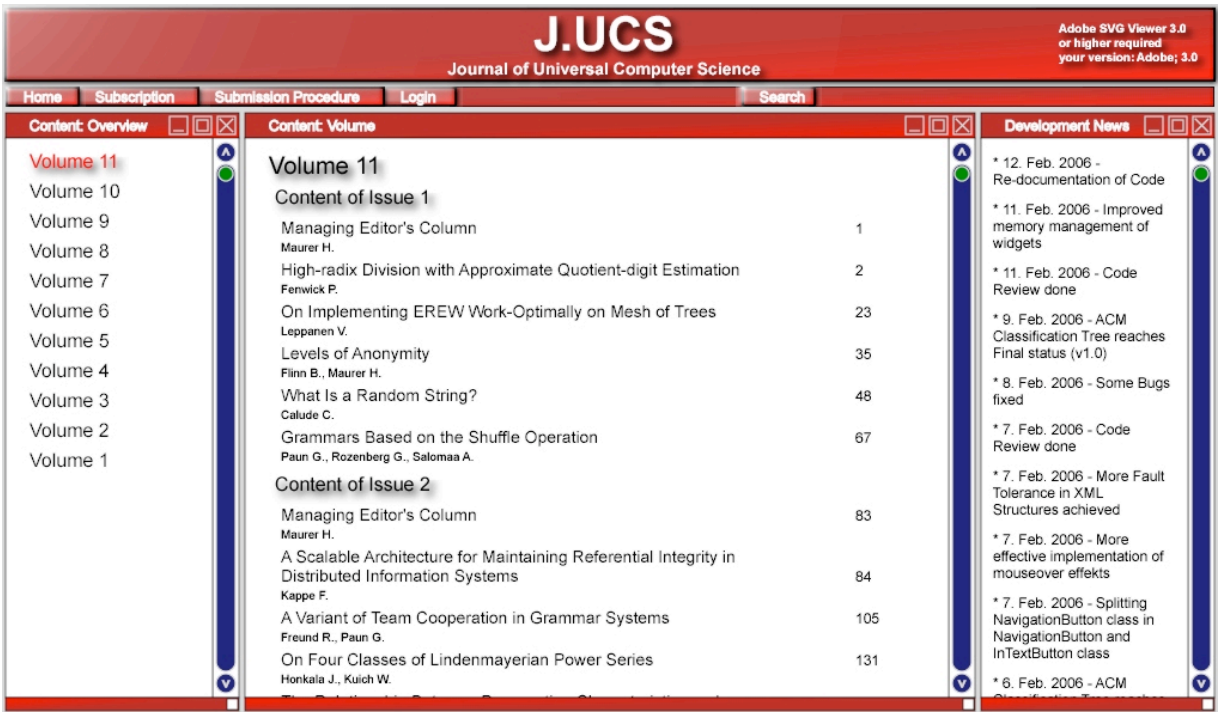

Figure 3: Navigating through the Digital Library

6 Results

As a result of our experience in using SVG as a GUI for a web-application we have developed a prototype for the Journal of Universal Computer Science. A very general view of the prototype is sketched in Fig. 3. The layout in the SVG-environment itself is very similar to the layout of the HTML interface. This makes it easier for users to switch to the new technology.

Since windowed information entities are well known to users we decided to use windowed components. It is possible to move these components around and to close some of them (e.g. if they are not important to the user). In Fig. 4 we see a view of a paper.

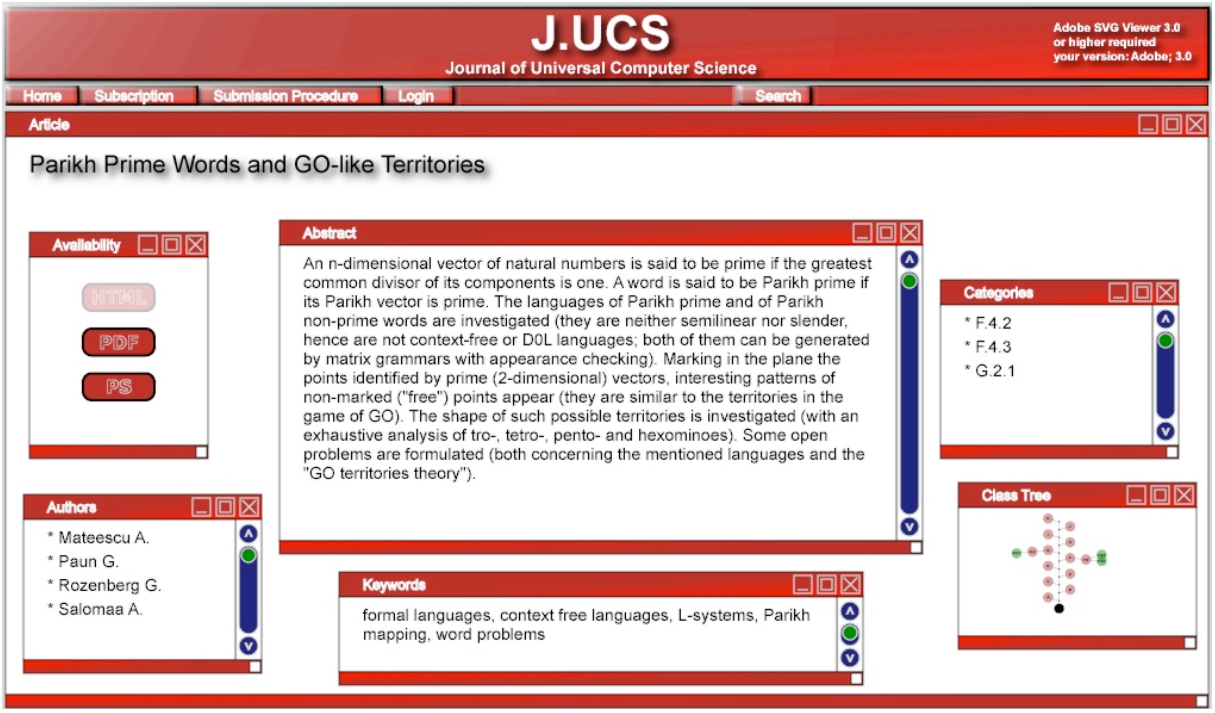

Figure 4: Visualized information about a Paper in J.UCS

This view looks - as already mentioned - very close to the HTML-view. Global navigation such as “Home” and “Search” are presented in the top-part of the page. “Local” navigation (such as “Fulltext in different electronic formats”) is located in the content part. Additionally to the elements visualized in the HTML-framework a classification tree is displayed. The classification-information is stored in the metadata-set of the paper-collection. The user may resize this classification-window and may rearrange single nodes of that tree. The category of the paper is visualized in a different color to make the relations more visible to the user. Mouse-over-effects visualize additional information about the related classification. A more detailed view of the classification tree can be found in Fig. 5.

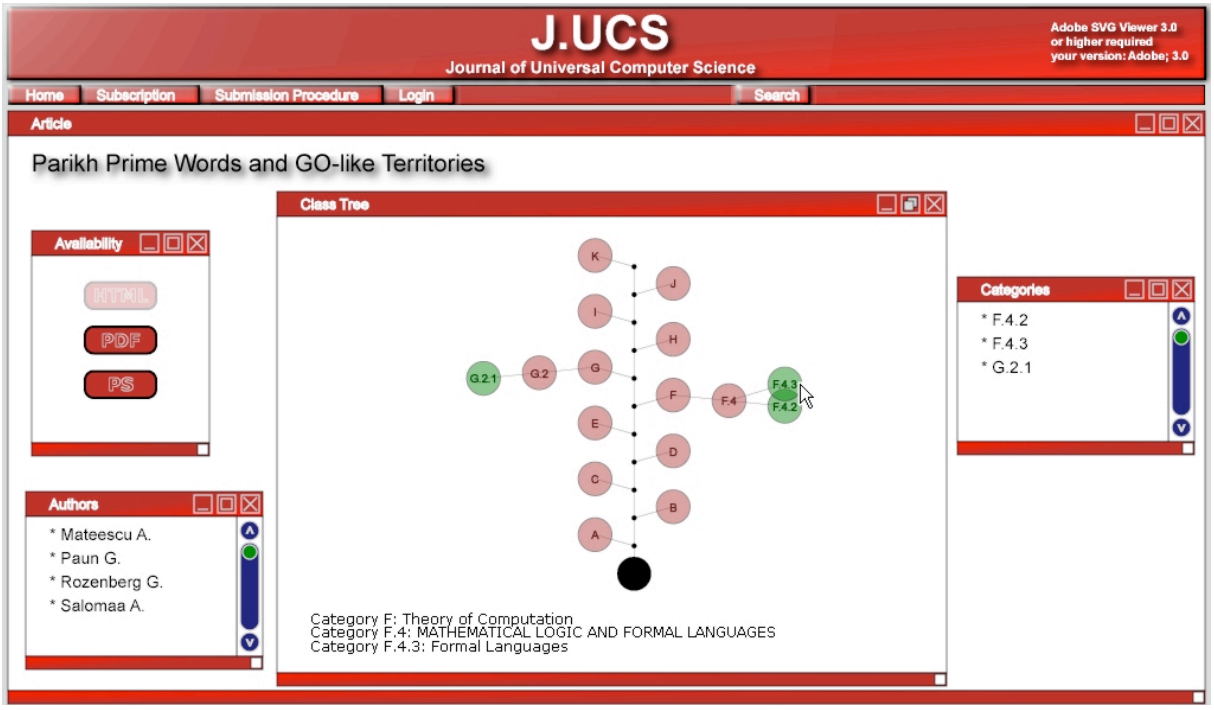

Figure 5: The Class Tree visualizing the ACM classification of the paper

The classification tree can also be used to navigate through the categories or to start a search inside the selected category. We think that this additional view of the categories of papers makes it very easy for users to browse the document collection. However, all the other information may be visualized using standard HTML technology.

7 Discussion

SVG is currently often used in cartographic applications. In this type of application many features of SVG are used in a very efficient way. Features such as text-alignment on a path (e.g. to name a street along the path of the street) and resizing of text-elements and vector graphic-elements are heavily used. The developed prototype shows some of the advantages of SVG in the context of GUIs for web-applications.

Animations and interaction with graphical elements are essential features in the GUI-development. Resizing of text and graphics, handling of text inside SVG documents are just some of the features we have worked on. The prototype showed that two-dimensional graphic elements are easy to describe by using standard XML-tools. Integrating existing content, as well as adding functionality to single elements is possible.
However, the prototype showed also some problems when working with this technology in that context: lack of development tools and integrated development environments, WYSIWYG-editors, debugging facilities, viewers, etc. Other deficits are in the current specification (SVG 1.1): access to data is not sufficiently specified. In our prototype we had to use a proprietary solution to access documents (which limits the usage of the application). This is surely not what one wants.

Another missing feature in the current specification (and therefore the implementation of the viewers) was the lack of automatic line-breaking. Since the specification does not describe how to break lines we had to implement JavaScript methods doing all the line-breaks. It is very obvious that this additional effort in formatting paragraphs should not be done by the developer of the application. To be honest: this is wasting time of the developer and also slows down performance when displaying pages to the user.
However, the new specification should solve at least this problem and should support developers in this aspect. Manufacturers of development environments and the open-source community will solve the problem with the lack of tools in the near future, when SVG becomes more and more popular.
A real standard for developing GUIs must also be developed. Best practice guides must be available to motivate developers of web-applications to use the technology. The prototype showed that SPARK is useable for the development of GUIs using SVG.

8 Conclusion

W3C published a recommendation for using XML to describe two-dimensional vector graphics. Many advantages of other graphic- and online-formats are covered in that recommendation. On the strength of the openness of SVG we can work in a very large application-domain: starting from simple graphical elements to complex applications. In this work we described how we handled SVG to build a graphical user interface for a Digital Library.

Using SVG-technology in this specific application-domain opens new features to users in a very intuitive way. Graphical representations of relations of objects are easily implemented using SVG. Colours, size of objects, etc. are visual attributes that can be useful to users of the system. Navigation in the system and browsing through the information-space are just two aspects that are easier to perform for users when using SVG instead of plain HTML. As an example we showed how to navigate in the ACM-classification tree. Using SVG-frameworks for the GUI makes it possible to even implement the whole web-application in JavaScript and SVG. Users have to install an SVG-viewer once to use the application.

Turing the project we have learned that it is very time consuming implementing SVG-applications without an integrated development environment. Debugging, profiling, etc. are possible using standard XML-tools, however it is difficult without dedicated and specialized tools. SVG in the current recommendation (1.1) lacks of features that are essential to developers. SVG 1.2 will solve most of the problems related to data-access. When more and more applications are available to users and different frameworks are available to developers than it will be much easier to use and spread this technology.

The current prototype will not be used in the production environment of J.UCS. However, we will follow the development of SVG 1.2 and we try to keep up with new features of that recommendation. We plan to implement one part of the project - the classification tree - using the new recommendation (SVG 1.2) and we will then provide users with that implementation.

References

[1] KROTTMAIER, Harald. Current and Future Features of Digital Journals. **In** *Proceedings of The First Eurasian Conference on Advances in Information and Communication Technology (EurAsia-ICT 2002).* 29-31 October, 2002, p. 495-502.

[2] FETTES, Alastair; LEWIS, Christopher; PETO, Chris; MANSFIELD, Philip. *SVG Programmers' Application Resource Kit.* 2004. http://spark.sourceforge.net/resources/downloads/index.html.

[3] FETTES, Alastair; MANSFIELD, Philip. *SVG-Based User Interface Framework*. 2004. http://www.svgopen.org/2004/papers/SPARK SVG Open.

[4] W3C Working Draft. Scalable Vector Graphics (SVG) 1.2. http://www.w3.org/TR/SVG12. 2005

[5] W3C Recommendation. Scalable Vector Graphics (SVG) 1.1 Specification. 2003. http://www.w3.org/TR/SVG11.

[6] Adobe SVG-Viewer. http://www.adobe.com