An XML-based Infrastructure to Enhance Collaborative Geographic Visual Analytics

Marc Kramis, Cedric Gabathuler, Sara Irina Fabrikant, and Marcel Waldvogel

ABSTRACT: We propose a new, streamlined, two-step geographic visual analytics (GVA) workflow for efficient data storage and access based on a native web XML database called TreeTank coupled with a Scalable Vector Graphics (SVG) graphical user interface for visualization. This new storage framework promises better scalability with rapidly growing datasets available on the Internet, while also reducing data access and updating delays for collaborative GVA environments. Both improve interactivity and flexibility from an end-user perspective. The proposed framework relies on a REST-based web interface providing scalable and spatio-temporal read-write access to complex spatio-temporal datasets of structured, semi-structured, or unstructured data. The clean separation of client and server at the HTTP web layer assures backward compatibility and better extensibility. We discuss the proposed framework and apply it on a prototype implementation employing world debt data. The excellent compression ratio of SVG as well as its fast delivery to end users are encourageing and suggest important steps have been made towards dynamic, highly interactive, and collaborative geovisual analytics environments.

KEYWORDS: Geographic visual analytics, data storage and access, interoperability, web interface, XML, SVG

Introduction

Geographic visual analytics (GVA) is a highly interdisciplinary research field, with tight links to different related disciplines, and having needs and interests in synthesizing information and deriving insights from massive, dynamic, ambiguous, and often heterogeneous data sources (Keim et al. 2006). The scientific objective of GVA is to understand how both individuals and teams carry out analytical reasoning and decision-making tasks based on complex information, and to use this understanding to develop and assess information and communication technologies for this purpose (MacEachren et al. 2006).

Increasing the sizes and complexities of data sets being collected, handled, and analyzed by visual analytics experts calls for new cross-disciplinary approaches (Andrienko et al. 2008). Efficient and effective storage and exchange of very large and complex distributed spatio-temporal databases is not only an important enabler for GVA, but also a research focus of the database research community within computer science. While previously large geographic datasets were typically of structured alpha-numerical nature (i.e., remote sensing images, census datasets, etc.) more recently GVA researchers have had to work with a multivariate mix of structured (relational) databases and increasingly semi-structured (e.g., XML-based) and unstructured (e.g., plain text) data sets, all readily available on the Internet.

A flexible and dynamic data storage and access infrastructure is especially needed when representing movement, dynamism, and change (Andrienko et al. 2008). Ideally, GV analysts should have efficient tools at hand for interactively access, rapidly modify, exchange in real-time, or generate entirely new representations on the fly from underlying

Marc Kramis, University of Konstanz, Department of Computer and Information Science, Box V 519, 78457 Konstanz, Germany. Tel: +49 7531 88-4734; Fax: +49 7531 88-3739. E-mail: <marc.kramis@uni-konstanz.de>. **Cedric Gabathuler,** University of Zurich, Department of Geography, Geographic Information Visualization & Analysis, Winterthurerstrasse 190, 8057 Zurich, Switzerland. Tel: +41 44 635-5151; Fax: +41 44 635-6848. E-mail: <cedric@geo.uzh.ch>. **Sara Irina Fabrikant,** University of Zurich, Department of Geography, Geographic Information Visualization & Analysis, Winterthurerstrasse 190, 8057 Zurich, Switzerland. Tel: +41 44 635-5150; Fax: +41 44 635-6848. E-mail: <sara.fabrikant@geo.uzh.ch>. **Marcel Waldvogel,** University of Konstanz, Department of Computer and Information Science, Box V 509, 78457 Konstanz, Germany. Tel: +49 7531 88-4948; Fax: +49 7531 88-3739. E-mail: <marcel.waldvogel@uni-konstanz.de>.

massive data sets, whenever the research context requires it.

Today, GVA user interfaces establish the necessary linkages between collected geographic datasets, data representations stored in databases, as well as with external (graphic) visualizations presented to a user which interact with internal (mental) representations. Ongoing technological developments provide continuously changing data types (i.e., from tracking devices, LBS, sensor networks, audio, etc.), which in turn require new data-handling structures for efficient GVA.

According to MacEachren and Kraak (2001), one of the challenges is to develop extensible methods and tools that enable the understanding of, and insights from, increasingly large and complex volumes of geospatial data that are becoming readily available. Scalability of GVA solutions has become one of the bottlenecks when dealing with massive databases.

Many different (server-side, client-side, hybrid) data-handling approaches are already available for Internet-based geographic information systems (GIS), and their goal is to improve data access performance (Chang and Park 2006). Each approach has its specific advantages and disadvantages with respect to data manipulation and management, user interactivity, and the distribution of server-side or client-side tasks (Chang and Park 2006; Yao and Zou 2008). Scalability and the provision of distributed collaboration varies significantly with each approach.

One of the main challenges for highly interactive and distributed GVA is the inherent potential for media breaks when dealing with distributed and diverse databases, thus reducing the potential for knowledge discovery. For example, knowledge might be disseminated through one media channel (such as written communication) in the form of emails or a journal article summarizing insights from a database that is no longer directly accessible from this particular media channel. Essentially, the media break is enforced by the underlying data infrastructure, as this infrastructure does not natively support the dynamic adaptation of large-scale data sets to various media channels. Each media break within a collaborative research context hinders knowledge discovery as it requires the (manual) conversion of data from one format to the next. The preparation, conversion, and reviewing steps all require time and significant computational resources when dealing with massive datasets. Consequently, real-time or interactive collaborations over a network are severely hindered.

This paper presents a data storage and a visual access framework capable of dealing with large-scale and frequently changing semi-structured (XML-based) spatio-temporal data sets being increasingly used in GVA research contexts. The proposed GVA infrastructure enables analysts to access and modify large and complex data sets and rapidly display these changes in response to user actions, thus enabling efficient and collaborative visual data exploration environments (Andrienko et al. 2008).

Specifically, we propose an XML-based infrastructure to reduce the potential number of media breaks within geographic visual analytics. Our infrastructure provides sound support to securely store and quickly access dynamically changing data, thereby providing adequate cognitive knowledge in a scalable, web-based, and collaboration-oriented way. We describe the underlying technology and provide a case study to demonstrate its benefits.

The rest of this article is organized as follows. The following section outlines the technological research context and related work. Then comes a description of the proposed GVA data storage and access infrastructure, followed by a section where we apply the implemented prototype to a case study. The next section discusses our research findings, highlighting opportunities and challenges of the proposed approach. This is followed by a concluding section which includes an outlook to future work.

Background

Recent developments foster the integration of data storage and display technologies in ways not possible before. The (well designed) web-based geovisualization display has become an interface to massive, complex and distributed databases that can support efficient information access and knowledge construction.

The Open GIS Consortium has initiated web mapping interoperability initiatives and specifications to develop interface specifications for geographic data (OGC 2002). This includes the Geography Markup Language (GML) encoding standard to express geographic features (OGC 2007), or the Web Feature Service (WFS) Implementation Specification for retrieving geographic features across the web (OGC 2005). In addition, geographic features stored in this fashion can be displayed using the Scalable Vector Graphics (SVG) format, an open standard developed by the world wide web Consortium (W3C) (Peng and Zhang

2004). SVG makes use of the eXtended Markup Language (XML) to describe two-dimensional geometric objects (points, lines, and polygons). In Neumann and Winter (2001)'s words, XML is the future core-technology for all upcoming web standards.

Peng and Zhang (2004) have outlined the role of GML, SVG, and WFS in building an Internet-based geographic information system (GIS). Open issues were in their opinion the compression of GML and SVG files, seen also as the easiest issues to solve. More complex open issues are the client-side SVG user interface and data processing tools to assist users as they interact with GML data. More recently, Yao and Zou (2008) highlighted interoperability challenges of Internet mapping tools based on the open source approach. A core challenge is the efficient transfer of data between relational and object-oriented databases. For example, widely used proprietary databases such as ESRI ArcSDE or Oracle Spatial store geospatial information in a long, binary data type in an unpublished format. To access these data for display with SVG, an SQL query is required. The traditional approach is to deliver the requested data in a Standard Open Format, e.g., an ESRI Generate File. An intermediate data conversion step is then required to generate the SVG document from the ESRI Generate File, before it can be presented to the user in the form of an easy-to-use graphical interface (Dunfey et al. 2006).

According to Neumann and Winter (2001), databases are easier to query or update while XML is perfect for data exchange and archiving.

SVG displays can be constructed directly out of (XML) database and be presented to a user for interactive geovisualization and visual analytical knowledge construction. SVG is optimized for graphic rendering on the web. Features such as vector display, animation, interactivity, transparency, graphic filter effects, shadows, lighting effects, and easy editing are all provided with SVG (Yao and Zou 2008). However, while SVG is eminently suitable for graphic content delivery by providing flexibility for user interactions (Neumann and Winter 2001), one should recognize the problem of missing topology for advanced spatial analysis and such limitations in cartographic symbolization as missing complex line styles.

Approach

We propose a web-based, flexible, and scalable GVA framework using native, XML-based data storage and back-end handling infrastructure coupled with SVG at the system–user interface. This GVA infrastructure provides analysts with highly interactive GVA tools to support complex data exploration and decision-making tasks. It includes flexible data depiction, high computer–user interaction, and collaboration over the web.

We favor SVG for our approach, as it allows for rapid system development and prototyping, provides fast response times for interactive query requests, and supports efficient data interoperability over networks (Yao and Zou 2008). Similarly to Yao and Zou (2008) and Dunfey et al. (2006), we expect that SVG will be supported natively in most if not all web browsers, and thus no extra plug-ins will be necessary.

We natively store SVG data in an XML-based database, even though other authors have argued against using SVG as basis for geovisualization (Yao and Zou 2008), because it is not suitable for securely and efficiently storing, managing, or delivering spatial data over the network. We argue that TreeTank solves such problems as secure and efficient storage, management, and network-based data delivery. Another XML-based language, the Geographic Markup Language, specifically targets geographic data. Fortunately, SVG and GML are highly compatible and can work in synergy. For example, Yao and Zou (2008) convert GML-based data to SVG before transmitting data to the client for display.

We employ the representational state transfer (REST) technology for queries to, and feature extraction from, our XML database. REST is a set of network architecture principles which outline how resources are defined and addressed. Practically speaking, REST defines a simple and scalable interface for exchanging resources over the Internet using the HTTP protocol. Each resource must be uniquely addressable through hypermedia links, meeting a universal syntax. A well defined and typically a small set of HTTP operations specifies how to proceed with the obtained resource. The basic operations are POST to create a resource, GET to read a resource, PUT to update a resource, and DELETE to remove a resource. The scalability and unquestioned expressiveness of REST makes it the interface of choice when it comes to handling large-scale SVG data on a network. The clean separation of client and server at the web layer (HTTP) allows both sides to be independently implemented, while drawing from state-of-the-art standardized web technologies such as, Java, Ruby on Rails, or Adobe Flex. In addition, REST is a bidirectional interface both for querying and modifying the requested resource (Fielding 2000).

Infrastructure

At the heart of our contribution lies the switch to a native XML database capable to directly store and emit fine-grained XML data. Unlike traditional relational databases, native XML databases do not store the XML data as character large objects (CLOB) and inherently know about the XML structure and XML nodes. The finer granularity allows answering complex queries and extracting the stored XML in a scalable fashion because there is no parsing and reconstruction as required with character large objects. In addition, most state-of-the-art native XML databases support modifications of the stored XML.

Our XML-based infrastructure consists of two components, i.e., the web interface called Temporal REST (Giannakaras and Kramis 2008), and the TreeTank storage manager (code name Idefix) as described by (Gruen et al. 2006). The two components are connected to implement a two-step workflow as follows:

- An **XQuery expression** is issued to TreeTank through Temporal REST;
- TreeTank returns **SVG** through Temporal REST.

In stark contrast to the traditional three-step workflow based on relational spatial databases, the intermediate data conversion step is eliminated, i.e., there is no need for converting such a standard open format as an ESRI Generate File into SVG. The eliminated intermediate data conversion step makes heavy use of CPU and IO, which contributes to large end-to-end delay, thus virtually inhibiting interactive Geographic Visual Analytics.

The two following subsections give an introductory overview of the involved technologies.

Temporal REST

While there exists a variety of solutions to access XML resources over the web, there is—to our knowledge—no generic and unified solution to conveniently access:

- The current revisions of the XML resource or any subset thereof;
- The full revision history of the XML resource or any subset thereof; and
- The full modification history of the XML resource or any subset thereof.

We decided to work with XML as a fine-grained tree of nodes and evolve this tree over time through user modifications. As such, we realize that we can access single nodes or whole sub-trees, i.e., XML fragments, within a temporal dimension in a unified, scalable, and robust way.

Only if we consider the entire life cycle of an XML resource, including the past revisions and the (transaction-based) modification history, will we get a complete idea of its true power. Notably, collaboration processes frequently involve asynchronous workflows. As such, the effectiveness of the workflow largely depends on the ability to highlight the modifications which took place during the last (or any past) step of the workflow.

We use Temporal REST with its related protocol message exchanges to generically implement our idea of exploiting web-based XML resources. Based on the Pareto principle, our proposal is simple enough for the average web application developer and at the same time it is extensible enough to be used with complex setups.

There are three different ways of accessing nodes and subtrees (XML fragments) in an XML resource. These include (1) the step-by-step tree navigation (XPath), (2) the query including joins and other complex expressions (XQuery), and (3) the ID-based random node access (DOM). Temporal REST supports all three and complements them with a temporal expression as described later. Note that XPath is a subset of XQuery.

XML IDs enable the user to tag the XML document and to quickly access the XML fragment. However, most XML nodes are not tagged with such an XML ID and hence not available for random access. We suggest that at the least all element nodes are tagged with a system-generated REST ID. Text nodes or attributes are accessible through their parent node. Other XML nodes such as comments or processing instructions may be tagged by the system on demand. One advantage of having the system do the REST ID tagging is that REST ID remains stable throughout subsequent revisions and modifications, i.e., a node or its modifications can be accessed irrespective of the revision or position in the tree. Another advantage is the guarantee of the existence of an ID. The system can make the REST IDs visible by tagging the serialized XML with REST ID attributes bound to the namespace of Temporal REST. Figure 1 shows how an example XML document is tagged with REST IDs.

Each insertion operation assigns unique immutable REST IDs to all new element nodes. This assignment is made by the back-end that stores the XML, and it does not affect any existing user-assigned XML IDs. REST IDs are numerical, and they are incrementally assigned starting at one. REST IDs do not necessarily need to be assigned

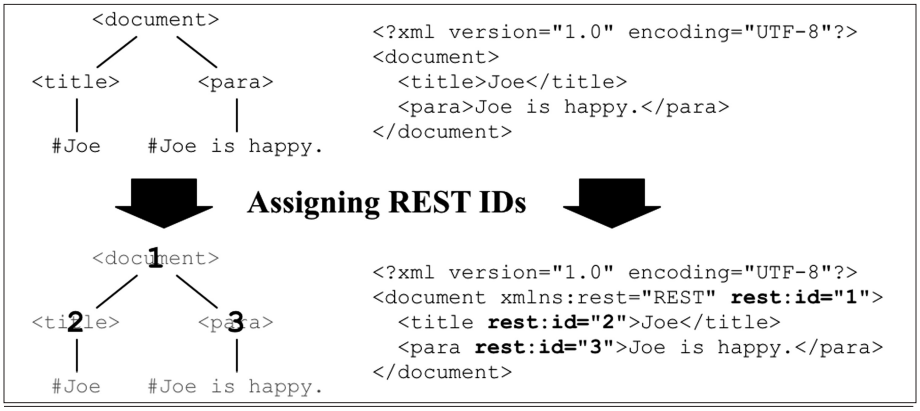

Figure 1. Each XML node gets its own ID.

in document order and they must not change once assigned to a node. In addition, we suggest that REST IDs not be re-used, so as to minimize confusion due to reassignments in future revisions. Since deletions are less frequent than insertions with most real-world workloads, the steadily shrinking space of available IDs is considered to be a negligible problem.

Each insertion, update, or deletion of an XML node results in a modification event. Each new revision event is assigned a timestamp, an author, and a comment. Temporal REST communicates modifications by encapsulating the modified node within an item element. This element contains the REST ID of the modified node as well as revision, time stamp, author, and comment information. As such, both the insertion and the deletion can be considered as setting a node to a new value. Deletion sets the node to the empty node. We opted for this approach for two reasons. First, we can streamline the transport of XML fragments and modifications within the XQuery data model, i.e., within a sequence of items. Second, the back-end can combine the storage of the modification event and the result of the modification.

The select operation allows the retrieval of a sequence of items as defined by XQuery. Each item either is an atomic value, or an XML node, or a modification event. The selection can be query-based or REST ID-based. Temporal REST will restrict the execution domain of both the query and the REST ID according to the temporal expression by either selecting a point in time or a time period. While a query may return a sequence of multiple items, an access solely based on a REST ID will return a sequence with at most one item. If the query and REST ID approach are combined, the query treats the node with the given REST ID as the root node of the query. The query-based approach makes it possible to add new query languages in the future and express complex queries, including operations such as full-text search or joins. The REST ID-based approach makes it possible to directly select an item with optimal performance because the system does not have to compile and optimize the query.

The temporal expression must be enclosed with round brackets '(' and ')' and contain a single point in time or a time period consisting of two points in time separated by a dash '-'. A point in time can be a revision number, an ISO date in short notation, i.e., without dashes or colons, or nothing. If no date or revision is provided, the last successfully committed revision is selected. Note that the ISO date in short notation is compliant with the specification of a URL. A single point in time will retrieve the XML fragments as they were at the given revision. The time period will retrieve the modifications between (and including) the two provided points in time in ascending or descending order. Leaving out the temporal expression automatically causes a fallback to the last successfully committed revision for backward compatibility. Table 1 shows the HTTP request and response required to either select a single point in time (Example 1) or a time period (Example 2).

A single node or a whole sub-tree can be inserted either as the first child of an existing node or as its right sibling. As such, the insert operation requires a query selecting a number of nodes or a REST ID besides the actual XML fragment to complete the insertion. During the insertion process, the back-end system will assign REST IDs as described above. Note that the insertion of an attribute must be made with the PUT operation which changes the whole node.

A single node can be replaced with or without the replacement of its sub-tree. Again, the updating operation requires a query to select a number of nodes to update or a REST ID. In addition, the actual updated XML fragment has to be provided. Restricting the effect of the update to the node (not effecting its sub-tree), allows the insertion of an attribute into an existing node without changing its whole sub-tree.

Whenever a node is deleted, the node and its sub-tree are purged from the system (but not from

the past revisions). The deletion operation requires a query or a REST ID to select the nodes to delete.

TreeTank

We tried to implement Temporal REST on top of existing open and closed source technologies. Unfortunately, it turned out that there was no file system, no relational database, and no native XML database to efficiently support all our requirements at once. All systems struggled with the combination of large-scale, heterogeneous, and revision-based data. In fact, it was possible to mimic the revisioning feature, i.e., to keep past revisions for given data. However, the resource consumption with respect to disk, memory, and CPU already exceeded the capabilities of state-of-the-art systems even for data sets far smaller than a single gigabyte. Eventually, we decided to implement our own system to overcome these limitations and called it TreeTank. Three systems mainly influenced our work on TreeTank. The ZFS (Sun Microsystems, Inc. 2004) file system handles transactions and snapshots but still operates at file-level granularity, which is far too coarse for small-grained XML data. The revision control system Mercurial brought along Revlog (Mackall 2006), a space-efficient method to store all past revisions—again only at file-level granularity. XPathAccelerator (Grust 2002) inspired the low-level XML encoding of TreeTank as it makes it possible to work efficiently with read-only large-scale heterogeneous data sets.

	HTTP Request	HTTP Response
1	GET http://../document/(1)?//para/text()	<?xml version='1'?> **<rest:response xmlns:rest='REST'>** **<rest:sequence rest:revision='1'>** **<rest:item>** Joe is happy. **</rest:item>** **</rest:sequence>** **</rest:response>**
1	GET http://../document/(2-3)	<?xml version='1'?> **<rest:response xmlns:rest='REST'>** **<rest:sequence>** **<rest:item rest:revision='2'>** <para **rest:id='3'**> Mike is happy. </para> **</rest:item>** **<rest:item** **rest:revision='3'** **rest:item='2'/>** **</rest:sequence>** **</rest:response>**

Table 1. Two example REST request and response pairs. Example 1 retrieves an XML sub-tree at revision one. Example 2 retrieves the modifications on the whole document during revisions two and three.

TreeTank is a native XML database designed to provide scalable read and write access to XML data. TreeTank concurrently allows multiple read transactions and a single write transaction each of which creates a new revision per transaction commit. TreeTank was designed to be secure and easy to maintain. The scalability of TreeTank results from the concurrent use of resources such as processing and storage units and from the design of the main internal data structure to store the XML tree.

The decision to only support a single write transaction at any time makes it possible to run any number of processes concurrently, accessing any past revisions or modifications. The newly modified data are clearly separated and only become visible after the last successful transactional commit to processes different from the write transaction process. If multiple users want to work on the same XML tree at the same time, a transaction manager is required to coordinate, i.e., sequentialize the changes, or a workflow has to be established stating clearly when each user is allowed to work and what he or she can do. Alternatively, a locking scheme has to be established which may follow an optimistic or pessimistic locking policy. However, it turns out, that in many real-world use cases, only a single user is working on a given part of the tree at any time, or that the natural workflow of a team working with XML data resolves modification conflicts before they even could appear.

The data model of TreeTank was intentionally chosen to be equal to the data model of Temporal REST (see Figure 2). Each XML resource, i.e., an SVG file, is bound to a session. The session is allowed to start transactions. Read-only transactions support two different selection modes. The first selection mode makes it possible to answer the question of what the data looked like at a given point in time. The second selection mode leads to an answer to the question of what changed between two different points in time. A write transaction can select the last successfully committed

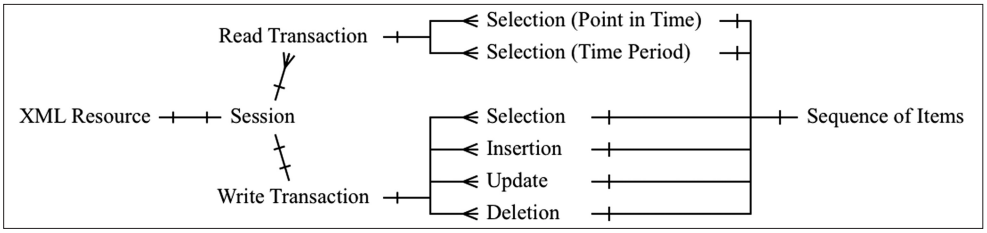

Figure 2. The data model of Temporal REST and TreeTank. Note that each item consists of an XML fragment, i.e., a value, an XML node, or a whole XML sub-tree.

point in time and modify it. Note that TreeTank currently does not support checkpoints within a single write transaction, i.e., the modifications on an XML tree are made persistent at once during the commit operation. A rollback can revert the XML tree to any past revision.

The data structure of TreeTank was optimized for updates. At most three directly related nodes must be updated whenever a single node or subtree is modified. Only the modified nodes are stored on disk in a compressed page. Note that traditional databases usually store the whole page (which may potentially contain dozens of nodes) even though only a single node may have changed. Still, care has to be taken that reads do not have to collect a huge number of scattered changes to reconstruct a single page. We opt to intermittently store a snapshot of the whole page to also support reads with reasonable performance. Compressing all pages, storing only the page modifications, and intermittently storing snapshots of the pages all help to reduce the storage requirements by one order of magnitude. As a result, TreeTank does not consume significantly more space and it can swiftly reconstruct any past state or modification.

Security is not a choice with TreeTank—it is always activated. Care was taken to implement only time-proven cryptographic primitives with sufficient key lengths and well chosen cryptographic modes so as not to create a weak link which could be attacked to break the whole system. TreeTank encrypts all compressed pages before they are stored on disk. This guarantees the confidentiality of the stored XML tree, even if the TreeTank files are exposed to the public or transferred through insecure networks. Besides the encryption, a strong message authentication code is derived from each compressed page and stored with a reference to the page. As each reference contains the message authentication code of all its children, the integrity and authenticity of the whole TreeTank can be verified recursively. The root message authentication code can be securely signed and further secured by an external secure time stamping mechanism, which also ensures that modifications cannot be denied. The availability of TreeTank can be guaranteed on the application level by a master–slave replication which consumes very little network bandwidth and is perfectly suited for geographically distributed operations. The master–slave setup ensures that all modifications applied to the master are synchronously or asynchronously propagated to the slave. The tight integration of security enables storage of sensitive data in the TreeTank. This is especially important because visualizations are usually based on large data sets collected from the internal operation of an organization or project and must not be exposed to the public.

Preliminary measurements on a state-of-the-art desktop computer show two significant advantages of TreeTank. First, it compresses the original XML data while storing it in its native data structure. Second, it enables a fast retrieval of the original XML. The promising preliminary results of the compression and time measurements for three SVG files of different sizes are as follows:

- The size of the TreeTank is up to ten times smaller than the original SVG file and TreeTank can deliver the original SVG data up to twenty times faster than a relational database with spatial extensions.
- The excellent compression ratio is due to the verbosity of SVG.
- The time of the data conversion step alone (excluding the time to retrieve the original data from the spatial database) takes much longer than the time required to retrieve the whole SVG from TreeTank.

Case Study

In this section, we provide a case study to demonstrate not only the feasibility but also the significant benefit a user can gain from our infrastructure. Most importantly, we want to build a mindset for designing and using our infrastructure because it is notably different from traditional workflows both on the technical and application levels. With our infrastructure, the user can organize and later modify the data in the XML tree, as he or she likes. He can mix

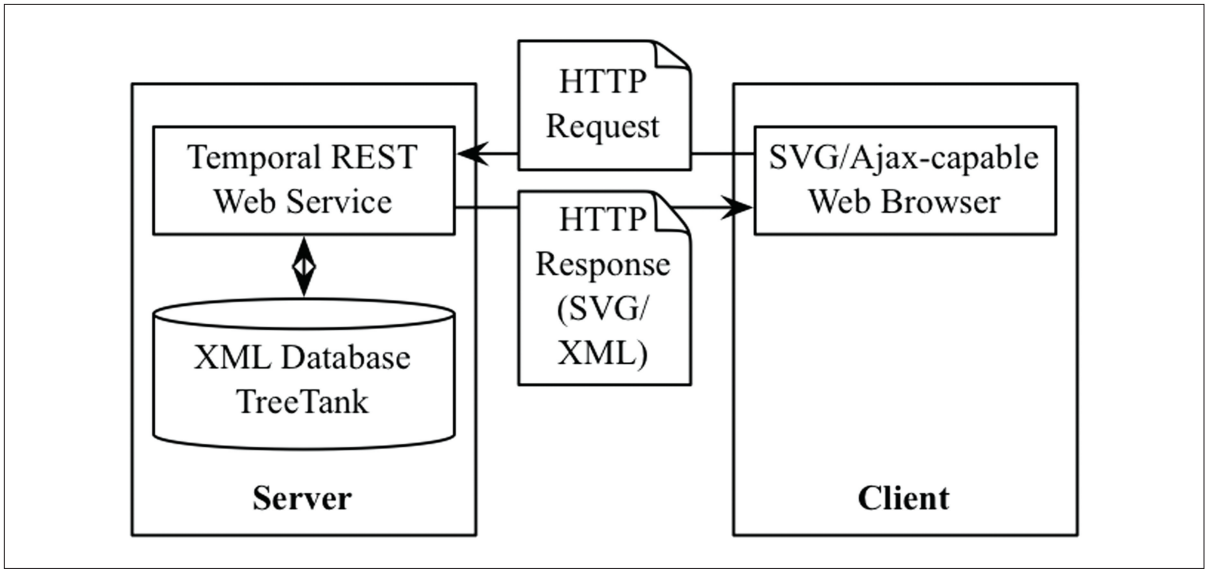

Figure 3. A typical setup of our XML-based infrastructure. Server and client exchange SVG/XML data with HTTP-based Ajax technology.

document-centric sub-trees containing information, e.g., in the OpenDocument format, with sub-trees compliant with ready-to-visualize SVG data, as well as data-centric statistical information. Figure 3 shows a typical setup of our XML-based infrastructure.

In this case study, we build an example TreeTank of gross external debt positions in U.S.$ per person. This information is available on a quarterly basis (Worldbank 2008) and perfectly suited to illustrating how a team can create sophisticated visualizations based on a set of statistical data. Four revisions of the visualization can be seen in Figure 4.

Note that the TreeTank is exposed to authorized users through a web service running Temporal REST. While we intentionally present a basic example, our infrastructure can deal with any large-scale heterogeneous data as long as the data can be transformed into XML.

The first step is to convert the Excel-based statistical information into a data-centric XML. This is a straightforward step and only required if the original data are not available as XML. The resulting XML can be directly imported into TreeTank by inserting the whole XML document through Temporal REST. We can now query Temporal REST to extract the whole document or any sub-tree therein.

For the second step, we need an SVG representation of the world with all countries. One can rely on open source SVG world maps or retrieve an individually configured world map from a traditional relational spatial database, depending on requirements. To keep the statistical data separate from the SVG data, we insert the new node *statistic* as the parent of the statistical XML data. Then, we insert a new node *geodata* as the right sibling of *statistic* and group the two nodes *statistic* and *geodata* under the third new node *example*. We then insert the whole SVG data under the node *geodata*. Hence, we can retrieve the plain statistical data by selecting the sub-tree rooted at *statistic* or visualize the world map within any SVG-enabled web browser by selecting the sub-tree rooted at *geodata*. To combine the statistical data with the visualization, we have to make sure that both sub-trees store the ISO country codes for each country. If this is not already the case, we can update each country in each sub-tree. Note that most SVG-based world maps will separately store an SVG path for each country.

Meanwhile, we created a set of revisions, each consisting of a Temporal REST modification request. At any time, we can retrieve an older revision or list the modifications applied to past revisions. This is convenient, if one wants to know what changed, e.g., in the sub-tree under *geodata*. It is also assuring to know, because one can revert the tree to a past revision if an unintended modification took place. At no time, data are overwritten or lost. Furthermore, the author of the changes can provide commit comments with each Temporal REST modification request to document his intentions and the evolution of the tree.

We prepare the visualization of statistical information by defining value ranges and color schemes for each value range. Then, we add the color information as an XML attribute to each element in

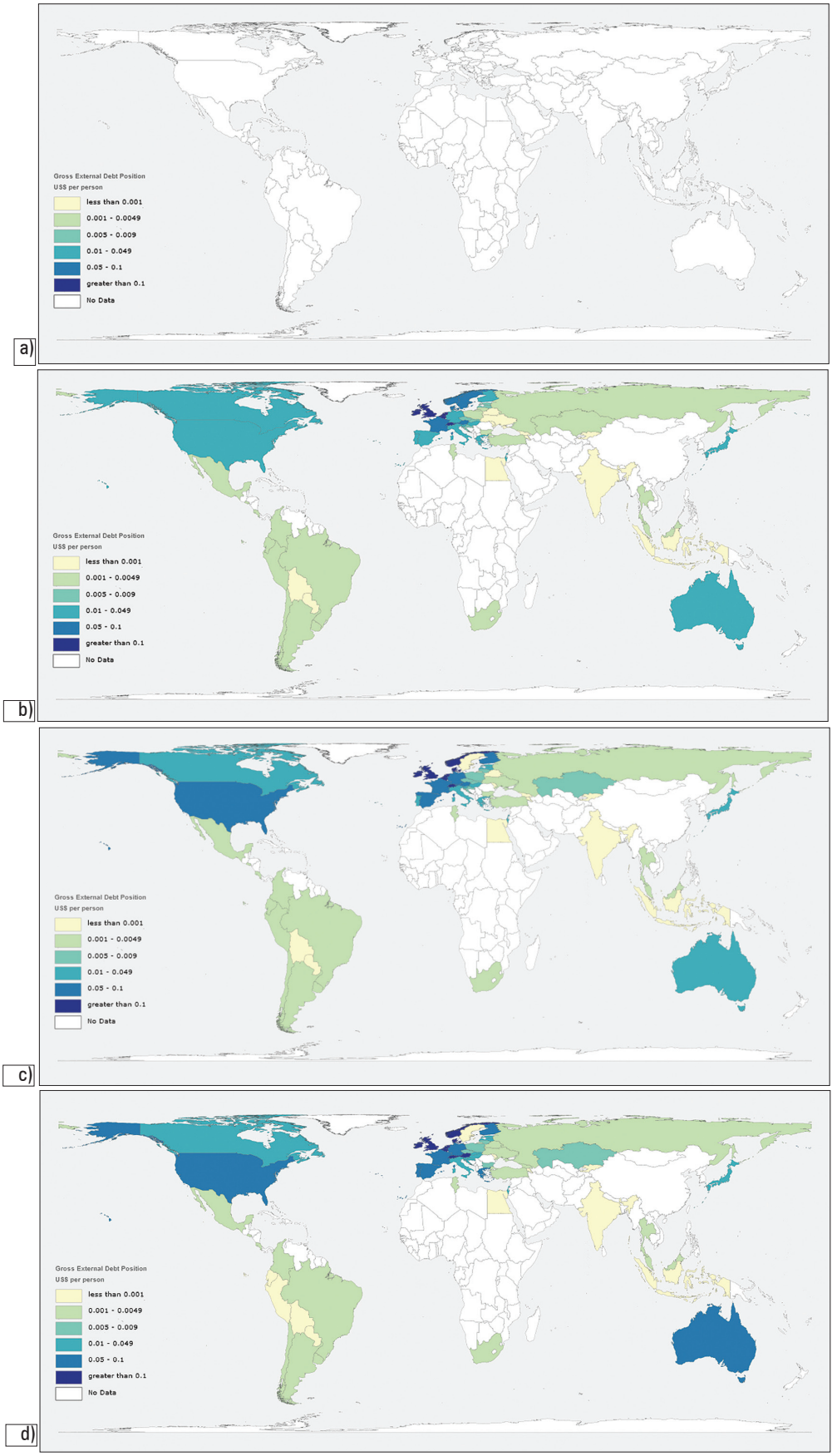

Figure 4. TreeTank of gross external debt. **a)** The SVG sub-tree with the map of the world and a description box. Figure **4b)** The gross external debt positions in U.S.$ per person for the year 2006; third quarter. Figures **4c)** and Figure **4d)** show the same information for the years 2007 and 2008; third quarter, respectively.

the *statistic* sub-tree based on the statistical value and make sure that the statistical information is grouped in sub-trees for each year and within the years for each quarter. Next, we add the SVG elements to the *geodata* sub-tree required to draw a box displaying the color scheme and value ranges. To better support layering in SVG, we group the SVG elements required to draw the box under the XML element *description* and then group the SVG path elements required to draw the world map under the XML element *worldmap*. This step helps to interactively enable or disable layers and can later be extended to support, e.g., layers containing water bodies, charts, or other GUI elements required for improved and convenient user interaction. To prepare the coloring of the countries according to the selected statistical data, we add the appropriate SVG color attribute to each path element. Finally, we add a SVG GUI element under *geodata*, which enables us to interactively select a quarter of the year.

The actual procedure to color the world map according to the selection can either be implemented with an XQuery expression issued through Temporal REST or with JavaScript on the client side. If XQuery is chosen, one must select both the *statistic* and the *geodata* sub-trees and then set the color attribute of the SVG path elements to the color attribute of the statistical data by joining them by country code. When JavaScript is preferred, both the *geodata* and the sub-tree containing the statistical information for the selected quarter have to be transferred to the client and then joined together by looping through all countries and setting their color to the color value found in the statistical data. Note that the statistical data can be reloaded efficiently and on demand with Ajax technology.

The main differences between the XQuery and the JavaScript variant is the location where work is done (i.e., on the client or the server side) and the amount of data that has to be transferred over the network. In the case of XQuery, the join is calculated on the server side for each request. Then the result is transferred to the client and immediately visualized. In the case of JavaScript, large amounts of data have to be transferred to the client for the first request in order to calculate and visualize the join. For later requests, only the new statistical data are transferred, joined, and visualized. Thus, JavaScript is the better choice if the workload consists of multiple selections for different quarters. However, note that current JavaScript runtime environments are so slow that the XQuery variant might be faster even though all data for the visualization have to be transferred for each request. This may change in the near future sd most JavaScript runtime environments are currently undergoing major rewrites to speed them up significantly.

An alternative to the method of joining pre-calculated persistent coloring information with the map is the purely dynamic calculation depending on the current user requirements. Again, the calculation can be performed on the server or on the client side, with the same advantages and limitations as noted for the join method.

Figure 5 gives an additional example of laying out GUI elements with SVG (including a sample chart). The GUI elements of interest are the revision slider, the map layer control, and the search field. All events are handled by JavaScript which uses Ajax technology to fetch missing data from the Temporal REST web service. The JavaScript itself is embedded in CDATA sections of the XML. In our view, this is not the most elegant way to store JavaScript in CDATA sections. However it is a straightforward and practical solution, which automatically guarantees the revision of the application code itself. There already are technologies such as the XML user interface language (XUL) or Adobe Flex which describe GUIs and their interactions based on pure XML. More work is however needed before these solutions can be included in off-the-shelf web browsers.

We have shown that the XML tree can be grown exactly according to the user's demand. All relevant data sources can gradually be integrated with TreeTank and then queried and further modified from within one single infrastructure. While the last paragraphs only considered a single user performing the modifications, we describe the collaboration of multiple users collectively working on the same TreeTank in the next paragraphs. Note that each user can modify the XML tree and add more statistical data or visualization elements as described before. As any professional publication or authoring workflow, it is important, however, that each user behaves according to a policy. With TreeTank and Temporal REST, this means that concurrent modifications have to be done in disjoint sub-trees.

While the current version of TreeTank does not provide a facility to enforce this behavior, it can be implemented technically on the application layer or non-technically in the organizational structure. We suggest a hierarchical responsibility delegation scheme, such that at any time, one author (person or process) is responsible for a given sub-

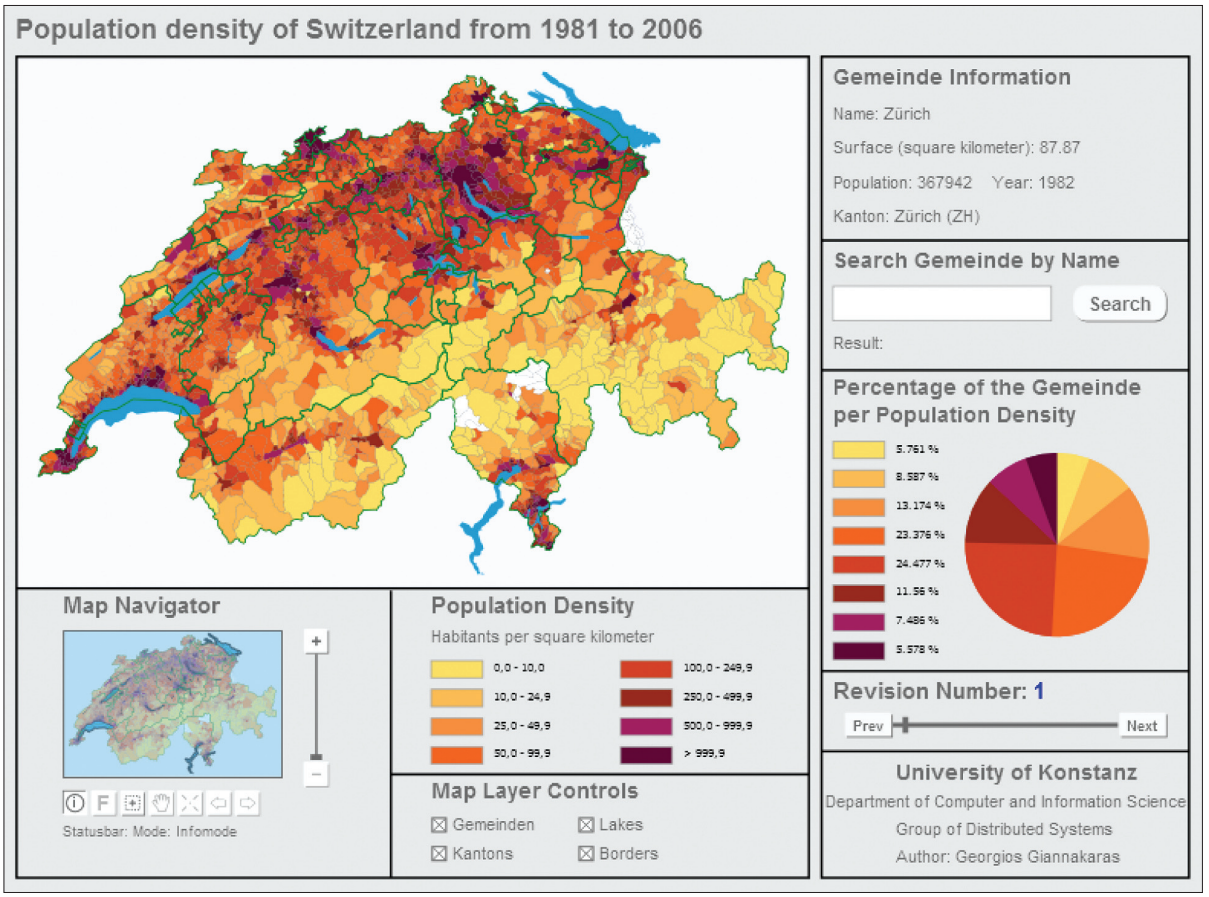

Figure 5. An example of a rich SVG GUI providing a chart and extended end-user input facilities. Note that this additional SVG sub-tree could be plugged-in seamlessly with the existing TreeTank.

tree unless he or she delegates a descendant to someone else with the option to revoke.

This scheme can be extended by a nonhierarchical access-control-list-based (ACL-based) scheme if required. To clarify the hierarchical responsibility delegation scheme, we imagine a situation where author A1 adds more statistical data each quarter, author A2 works on the SVG-based GUI and color schemes (Brewer 1994), and author A3 links the visualizations to scatter plots or other statistical graphics (Andrienko and Andrienko 1999). One possible hierarchical delegation then is as follows. The owner of the TreeTank delegates *statistics* to A1 and *geodata* to A2. A2 creates a new node *charts* and delegates it to A3. Then, all three authors concurrently modify the tree and will never cause isolation conflicts such as lost writes or dirty reads as they are stated in the ACID model, which is one of the oldest and most important concepts of database theory. Lost writes are prohibited by design because each author works in his responsibility domain, i.e., a dedicated sub-tree. Dirty reads are avoided because each author will only see successfully committed changes and has the option to query the tree as it was at a given, fixed revision. Whenever the user wants to switch to a newer revision, he or she first checks for modifications on the sub-trees of interest and whether they impact his own work, e.g., introduce an inconsistency because the color attributes were dropped. Finally the user can adapt his or her part of the tree to the modifications.

Discussion

The findings from the case study open a wealth of opportunities for the end-user as well as an array of research challenges. The immediate benefit of our XML-based infrastructure is the very efficient use of processing and storage resources. Much more user requests can be handled per time unit, and the degree of interactivity is dramatically improved as the user actions are no longer a matter of minutes but seconds. Both throughput and interactivity are essential

for collaboration-oriented environments where end-users are used to interact in an asynchronous as well as a synchronous fashion. The support for the evolutionary growth of tree (XML) data structures as well as the ability to store and query statistical and SVG data, side-by-side, help to reduce unnecessary media breaks which hinder the dissemination of (visually) discovered knowledge.

The research challenges are manifold. One challenge is to find and categorize tree structure and tree design patterns. Our infrastructure makes it possible to store huge amounts of unstructured data in a single TreeTank. Without patterns, the TreeTank could end up being a junk room where everything is contained but rarely something can be found in time. Hand in hand with the patterns comes the question of how best to organize and manage the concurrent access of multiple users assuming changing roles. In our case study, we suggested an organization form natural for tree-based data structures. But there may be other more efficient solutions. As with the tree structure and tree design patterns, the collaboration-oriented (authoring) workflows have to be collected, categorized, implemented, and tested with real teams. From a technical point of view, the challenge arises to integrate various indices with TreeTank to speed up specialized queries such as full text queries or spatial queries on rasterized data. While the server side can be further speeded up with the help of indices, the client side GUI and JavaScript environments still need to be revised to unleash the processing power of modern desktop or notebook computers. The GUI functionality of browsers and SVG plug-ins is not yet on par with native applications. Even the extensive use of Ajax and JavaScript does not hide the current shortcomings.

The case study made the assumption that there are multiple users but only one single TreeTank. When multiple teams concurrently grow their data structures in independent TreeTanks, the issue is how all these distributed TreeTanks can be integrated into one unified storage. While our infrastructure solves this by integrating different data sets into one tree, it does not yet provide support for integrating multiple trees into a forest.

Conclusion

We propose a new streamlined two-step GVA workflow for efficient data storage and access based on our native web-based XML database TreeTank and couple it with an SVG graphical user interface for visualization. Not only does our XML-based infrastructure substantially reduce access delays due to the elimination of intermediary data format conversion steps. Rather, it extends the user's options by providing significantly better scalability, inherent data security, and, most importantly, the ability to collaboratively work in GVA environments thanks to optimized update support. With up to twenty times shorter data access delays and up to one tenth of the traditional storage requirements, our infrastructure improves interactivity and flexibility from an end-user perspective.

Furthermore, our infrastructure suggests a paradigm shift leaving behind dispersed disconnected data sets and media breaks and introduces a tightly integrated unified storage for complex spatio-temporal datasets of structured, semi-structured, or unstructured data. The clean separation of client and server at the HTTP web layer assures backward compatibility and better extensibility. Future work will focus on fully implementing the latest XML query facilities such as XQuery, XQuery Update, and XQuery Full Text to give the end-user state-of-the-art tools with which to query large-scale data sets. Especially the full-text feature will further improve the value of our infrastructure for the collaboration-oriented end-user because he or she can freely search in all comments and documents stored along with the spatio-temporal data. We also plan to investigate how to most efficiently distribute TreeTank for even better scalability.

ACKNOWLEDGMENTS

This work was supported by DFG Research Training Group GK-1042 "Explorative Analysis and Visualization of Large Information Spaces" at University of Konstanz in Germany. We are grateful to Georgios Giannakaras who created a fully working case study as well as the world map figures during his master thesis at the University of Konstanz.

REFERENCES

Andrienko, G.L., and N.V. Andrienko. 1999. Interactive maps for visual data exploration. *International Journal of Geographical Information Science* 13(4): 355-74.

Andrienko, G.L., N.V. Andrienko, J. Dykes, S.I. Fabrikant, and M. Wachowicz. 2008. Geovisualization of dynamics, movement and change: Key issues and developing approaches in visualization research. *Information Visualization* 7(3): 173-80.

Brewer, C.A. 1994. *Color use guidelines for mapping and visualization*. In: MacEachren, A.M., and D.R.F. Taylor

(eds). Visualization in Modern Cartography, Tarrytown, New York:Elsevier Science Inc. pp. 123-47.

Chang, Y.S., and H.D. Park. 2006. XML web service-based development model for GIS applications. *International Journal of Geographical Information Science* 20(4): 371-99.

Dunfey, R.I., B.M. Gittings, and J.K. Batcheller. 2006. Towards an open architecture for vector GIS. *Computers & Geosciences* 32: 1720-32.

Fielding, R.T. 2000. Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine, California.

Giannakaras, G., and M. Kramis. 2008. Temporal REST—How to really exploit XML. *IADIS International Conference WWW/Internet*, Freiburg, Germany.

Gruen, C., A. Holupirek, M. Kramis, M. Scholl, and M. Waldvogel. 2006. Pushing XPath accelerator to its limits. In: *Proceedings of the First International Workshop on Performance and Evaluation of Data Management Systems (EXPDB 2006)*, Chicago, Illinois, USA.

Grust, T. 2002. Accelerating XPath location steps. In: *Proceedings of the 2002 ACM SIGMOD international conference on management of data*, Madison, Wisconsin. pp. 109-20.

Keim, D.A., F. Mansmann, J. Schneidewind, and H. Ziegler. 2006. Challenges in visual data analysis. In: *Information Visualization (IV'06), Tenth International Conference on Information Visualisation*, London, England. pp. 9-16.

Mackall, M. 2006. Towards a better SCM: Revlog and mercurial. In: *Ottawa Linux Symposium*, Ottawa, Ontario, Canada.

MacEachren, A., and M.J. Kraak. 2001. Research challenges in geovisualization. *Cartography and Geographic Information Science* 28(1): 3-12.

MacEachren, A., G. Cai, M. McNeese, R. Sharma, and S. Fuhrmann. 2006. GeoCollaboration crisis management, designing technologies to meet real-world needs. In: *Proceedings of the 2006 International Conference on Digital Government Research*, volume 151 of ACM International Conference Proceedings Series. New York, New York:ACM Press. pp. 71-72.

Neumann, A., and A. Winter. 2001. Time for SVG-towards high-quality interactive web-maps. In: *Proceedings of the 20th International Cartographic Conference*, Beijing, China. pp. 2349-62.

OGC. 2002. Overview of OGC's interoperability program. [http://portal.opengeospatial.org/files/?artifact_id=6196; accessed November 10, 2008].

OGC. 2005. Web feature service implementation specification. [http://portal.opengeospatial.org/files/?artifact_id=8339; accessed November 10, 2008].

OGC. 2007. OpenGIS geography markup language (GML) encoding standard. [http://www.opengeospatial.org/standards/gml; accessed November 10, 2008].

Peng, Z.R., and C. Zhang. 2004. The roles of geography markup language (GML), scalable vector graphics (SVG), and web feature service (WFS) specifications in the development of Internet geographic information systems (GIS). *Journal of Geographical Systems* 6(2): 95-116.

Sun Microsystems, Inc. 2004. ZFS: The last word in file systems. [http://www.sun.com/2004-0914/feature/; accessed November 10, 2008].

Worldbank. 2008. The Worldbank data & research. [http://econ.worldbank.com; accessed November 10, 2008].

Yao, X., and L. Zou. 2008. Interoperable internet mapping: An open source approach. *Cartography and Geographic Information Science* 35(4): 279-93.