Interactive Network Management Visualization with SVG and AJAX

Athanasios Douitsis and Dimitrios Kalogeras
– National Technical University of Athens, Greece

ABSTRACT

The area of visualization has always been one of the most attractive sections of network management technology. Successful management tools must not only fulfill objective management needs but also be aesthetically appealing. Consequently, the seemingly mundane task of presenting information to the user has become almost a true art. The application proposed in this paper is a vehicle for the presentation of network management data using interactive graphs. By using the Scalable Vector Graphics markup language (SVG) [1] and Asynchronous Javascript and XML (AJAX) [2], it strives to aid the rapid development of visually impressive management applications that are accessible through the use of a web browser. These highly interactive and versatile applications can respond to user actions and present data which is organized into layers and is retrieved and refreshed on demand. The data of these layers is generated by network management tools that plug in to the application through the use of a modular framework.

Motivation and Problem Statement

The Need for Abstract Visualization

Visualization of data has always been important in many areas of human knowledge and engineering, as it allows people to perceive information in more efficient ways which, in turn, can expedite the learning process and help understand and deal with relevant problems more efficiently. Many network management tools resort to various forms of visualization to depict the topology of a computer network and the relevant information concerning it. Unfortunately, representation of additional information is often cumbersome due to inefficient visual network topology abstraction.

Furthermore, most management tools tend to use inappropriate ways of producing their visual presentations, resulting in increased development effort and doubtful results from the user perspective. This is because most graphics generation libraries and APIs are too much low-level, which increases the burden of implementing the visualization part and consequently fail to abstract the production of visually rich network representations.

The Need for Interactivity

It is also generally recognized that a large part of the usefulness of a network management tool lies in its interactivity. The element of interactivity is essentially defined as the ability to dynamically change graphical representations and respond to user actions. In fact, interactivity is the very element that generally makes the difference between a real usable application and a mere depiction. Again, the development difficulty lies in the fact that the implementation of interactivity incorporates a great deal of unnecessary details and is generally cumbersome. So, a framework that can help easily produce highly interactive visual network management applications, will naturally contribute heavily to its success from both the perspective of the developers and the users.

Underlying Technologies and Dominant Trends on Network Management Tools

Depiction Of Networks As Graphs

Visualization of networks has been an area directly connected with the visualization of graphs. This is mainly because of the similarity between computer networks and graphs. In fact, it can be argued that one of the most efficient ways to present information on the status of a computer network to humans, is through a two-dimensional graph where the elements of the network (like routers and switches) are depicted as vertices and the connections between them as edges. Essentially all relevant network management tools today use this approach in one way or another, to provide a representation of real computer networks. Clearly, depiction of networks as graphs is the most intuitive choice for most purposes.

Separation of Functions

One other dominant trend among network management tools is the distinction between the presentation layer and the instrumentation layer. This distinction is highly beneficial as it allows to focus on the individual problems and solve them separately. One added benefit is that, if the model of presentation is well thought out, an interface mechanism can be created which can be potentially reused in many different network management tools. This is relevant to establishing a well defined presentation mechanism to be used by software to build interfaces easily.

Application Interface Technology

A question that has been posed many times in the past is which approach is the most suitable for applications to build a rich client interface, which is also connected with the element of interactivity. It should be noted that this question not only concerned network management tools, but applications of any kind. During the late nineties, it was thought that rich clients that were specifically developed for their distinct application, were the way to go. But with the proliferation of the Web, people soon realized that it is in fact more efficient (and easy) to develop applications that present their interface through a browser. During the last years, with the advent of Asynchronus Javascript and XML (AJAX) techniques, it was proved that web based applications can provide the same experience like any native rich client application. A large portion of the applications of tomorrow will be web based.

Regardless of the way the application interface is generated, in order to be used it must be made available to the user. Among popular network monitoring applications, some of the most common approaches to transporting the user interface to the user are:

1. The client application code is actually run in the server and the user interface is transported to the user terminal by using a specialized client-server protocol such as the X-Window System [3], VNC [4], ICA [5], RDP [6], SunRay [7], etc. This method is very powerful because it can transport a window or an entire desktop anywhere, but its use has declined because it places high loads on the server, it has several security disadvantages, it is cumbersome to deploy and use (special clients, permissions, etc.) and it may have gargantuan bandwidth requirements in order to operate in a speedy manner.
2. The client application is presented through a web browser, which in turn loads a presentation and management Java applet [8] that connects to the server using some special management protocol. This method is very flexible because the presentation and interface capabilities of the Java applets are really powerful and the protocol is specially designed for this particular application. However, the usage of a special client-server protocol for this purpose has other implications. On the server side, the protocol handler will typically have to be implemented from scratch, which introduces additional costs and adds security concerns (from having yet another protocol). Even more, on the client side the Java applet will have to carry classes that implement that special protocol, a fact that increases its size, makes it difficult to extend and more susceptible to bugs.
3. The client application is merely a web browser and the graphical images are produced on the server side. In this case, interactivity is very difficult to implement. On the server side, it requires the production of multiple images, which places additional burden on the servers. On the client side, it requires elaborate techniques using image maps which are difficult to produce and manipulate.

Generation of Graphics

To address the design requirements that are relevant to the graphical representation of graphs, a powerful way to create graphics inside a web browser is required. Usually, when a custom graphical image must be provided to the user through an application, the image is typically generated by the server on the fly using appropriate graphical libraries (like GD [9]) and fetched to the user. This approach is successfully used in our Multicast Weathermap [10] to generate images of network traffic over a geographical map. However, in many problems, the requirements for interactivity and dynamic manipulation makes the usage of a server generated static images inappropriate. It could be argued that, although server side graphics today constitute the vast majority of solutions today, there is clear indication that this may change in the future.

Solutions where the graphics are generated on the client side and can be manipulated dynamically include VML [11], Adobe Flash [12], Java applets and SVG. Of these options, the usage of SVG is the most appealing from a development perspective as:

- SVG is a W3C specification and the markup language is still being actively developed through new revisions.
- SVG uses an event model and interface to its DOM similar to ordinary web applications, with which there is already great familiarity among developers.
- There are already many browser based implementations available, like:
  - The Adobe SVG browser plugin [13].
  - Opera's implementation embedded inside the Opera browser [14].
  - Mozilla's implementation embedded inside the Mozilla Firefox browser [15].
  - The Apache Batik [16] project which can be used through Java Web Start [17] by a browser that launches the application. This is more cumbersome and is very rarely used.

Modularity

Lastly, today's modern networks are far more complex than yesterday's, and include a host of diverse technologies such as Multicast, IPv6, MPLS, etc., each one with its own requirements and peculiarities. It is unlikely that a giant monolithic management tool can be created which can be on top of all these aspects of the network. The present diversity of network management tools today can only serve as a proof to that assumption. In many cases, it is helpful

to use a modular approach, where many tools exist, each one is specialized in its own specific domain, and they are all using a unified presentation layer.

Design and Architecture

Design Goals

Based on the facts presented so far, the creation of a browser based tool that can be used as a generalized presentation layer for network management applications seems like a solution that can prove quite useful. The design goals outlining the characteristics of such a tool are:

- Usage of the tool through a web browser. As mentioned earlier, the delivery of the application through the use of a web browser is a practice that rapidly becomes commonplace, which easily justifies this choice.
- Representation of rich visual representations of graphs, allowing arbitrary usage of colors, styles, images and other artifacts.
- Graphical representations that are dynamically modifiable during application execution.
- Definition of arbitrary rules of interaction of the graphical elements with the user. This will allow customization to the point where the tool becomes truly intuitive to use.
- Extensibility. Individual presentations may require special capabilities which (at least in theory) should be relatively easy to implement with minimal modification.
- Well defined and simple API. This will make it easy to create new network management tools using the presentation application.
- Speed. The resulting application should be as light as possible (both on the client and on the server side).
- Security is also of great importance. Unauthorized usage should not be able to easily compromise the security of the system.

General Overview

In our proposed architecture (see Figure 1), the Network Management Station (NMS) collects data from the managed nodes in the network, processes it and makes it available for remote clients to connect and retrieve it. In this client-server approach, the client application is actually an SVG enabled web browser that similarly connects to the server periodically or on-demand and retrieves information server. The server on the other hand is the NMS which will typically do the collection from the managed nodes using protocols such as SNMP [18]. That way, administrative access to the network is assigned only to the NMS while the graphical rendering will be delegated completely to the client browser. Use of the client can of course be carried out from virtually anywhere.

The client application itself is an SVG document that contains code but is otherwise devoid of actual graphic content. To launch the application, the user typically has to navigate to the specific URL of the NMS server where this SVG document resides. The Javascript [19] code that is embedded inside the document is actually the implementation of the network management presentation application. Its purpose is, upon certain events, to retrieve management data from

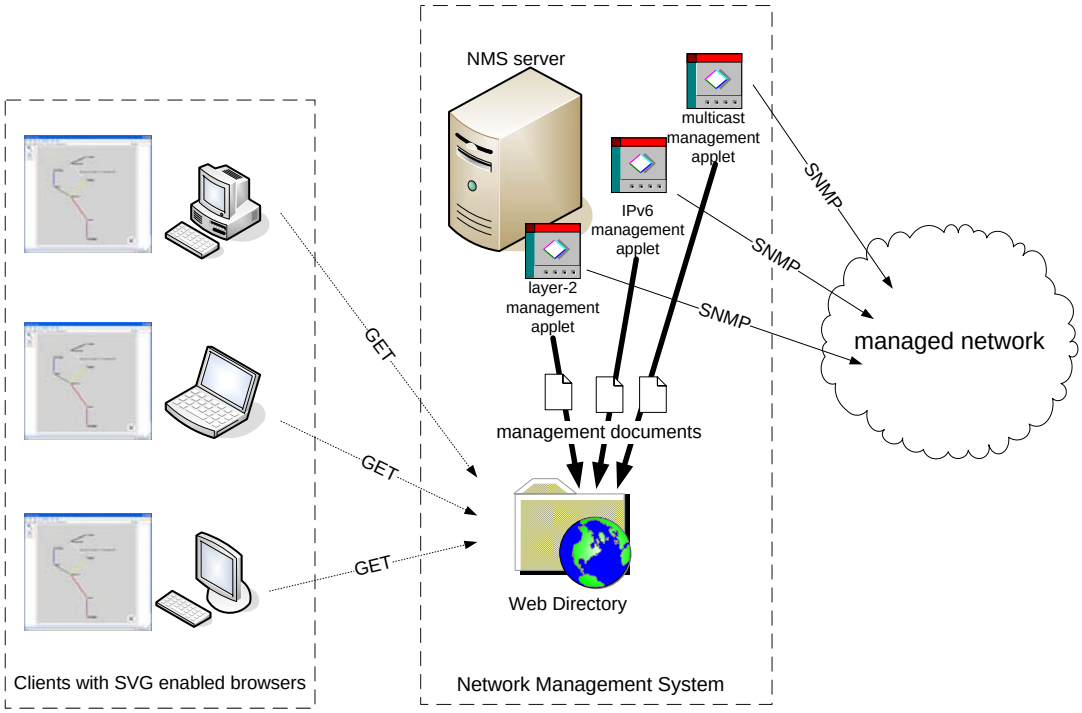

Figure 1: Architectural overview.

the NMS server and accordingly create, modify or delete graphical elements inside a page. The creation, deletion and modification of the SVG elements are done by directly accessing the SVG DOM. Furthermore, the events to which the application is designed to respond to, are:

1. Initial loading of the document. When the user first launches the application by navigating to its URL, the SVG document is empty. Responding to the event of its creation the application must retrieve all required data and create the initial graph.
2. Periodic updates. To make sure that the picture presented to the user is up to date, the application periodically updates the network graph based on fresh data.
3. User interaction. Responding to events generated by the user, the application may have to retrieve additional data and make the appropriate modifications to the graph.

If the content used to draw the graphical elements of the network was to be included inside the initial SVG document, the server would obviously have to dynamically generate this content and send it to the user. Although this is the most common solution, we believe that by supplying an initial document without graphical content and having it populate itself by querying the NMS is a much more elegant solution that can additionally scale up pretty well. That way, the client document that implements the client application is completely static, meaning that it does not need to be generated dynamically at all. It just simply needs to be placed somewhere where the client browser can get it.

Management Data Hierarchy

The management data that must be retrieved by the application is organized into XML documents that are made available by the NMS server to the client applications through HTTP. That way, the client applications can retrieve data by using the XMLHttpRequest [20] class, which is the typical way in which interactive AJAX applications work. The XML management data documents belong to following categories:

1. The **configuration document** that describes the general application configuration, mainly the HTTP locations of the other categories of XML documents to which actual data is contained. This is the equivalent of a configuration file for a conventional application. Upon its retrieval, it instructs the application to create multiple visualization overlays (described later) and supplies the required information to create and draw them. The configuration document is structured in such a way that a hierarchy between the visualization overlays can be described. This hierarchy has a double purpose, on the one hand to define the relative z-axis order of rendering of the overlays and on the other hand to present the user with a structured choice of management layers.
2. Documents that describe the nodes and their positions. Those documents are denoted as **node topology description** documents. A node topology description document usually contains all the managed nodes of the network (routers, switches, etc.) and their coordinates inside the SVG page. Especially for large numbers of nodes, the topographical layout of the nodes can be computationally intensive and cannot be carried out on the client side. Instead, the layout is transmitted to the client through this document along with the catalog of managed nodes. The coordinates of each node are always defined on the NMS server side and cannot be altered by the client user. The NMS server administrator can define the coordinates of each node by hand, which can be very helpful in cases where the nodes are relatively simple and must be drawn on top of a geographical map image. Alternatively, a program which is specialized in graph layout, like GraphViz [21], can be used to fully automate the computation of node coordinates in cases where a large number of nodes exist. A typical case where this is useful, is for large networks whose layout is complex and changes frequently, like a large campus LAN.
3. Documents that describe actual network management data and are denoted as management overlay documents. Each one of these documents refers to a specific domain of management information, for example there may be a document that provides the topology, status and traffic information of multicast inside a network, while another document may provide IPv6 traffic information, etc. Since the coordinates of each network node are already decided by the node topology description document, the management overlay documents define arrows, labels and other graphical elements that use the predefined coordinates of the nodes to be laid out. For example, for a network that is comprised of routers A and B, an overlay document can contain an element that instructs the client application to draw an arrow representing a link pointing from A to B. Obviously the overlay document need not bother itself with the definition and coordinates of A and B as these have been taken care of previously by the node topology description document. Overlay documents also contain interaction information as will be explained later.

It is also possible that some management overlay documents can be loaded on-demand, in response to user interaction. The pointers to the locations of these overlays are provided indirectly by other overlays. For example, when the user hovers over a network link in the graph of a specific overlay, the application may

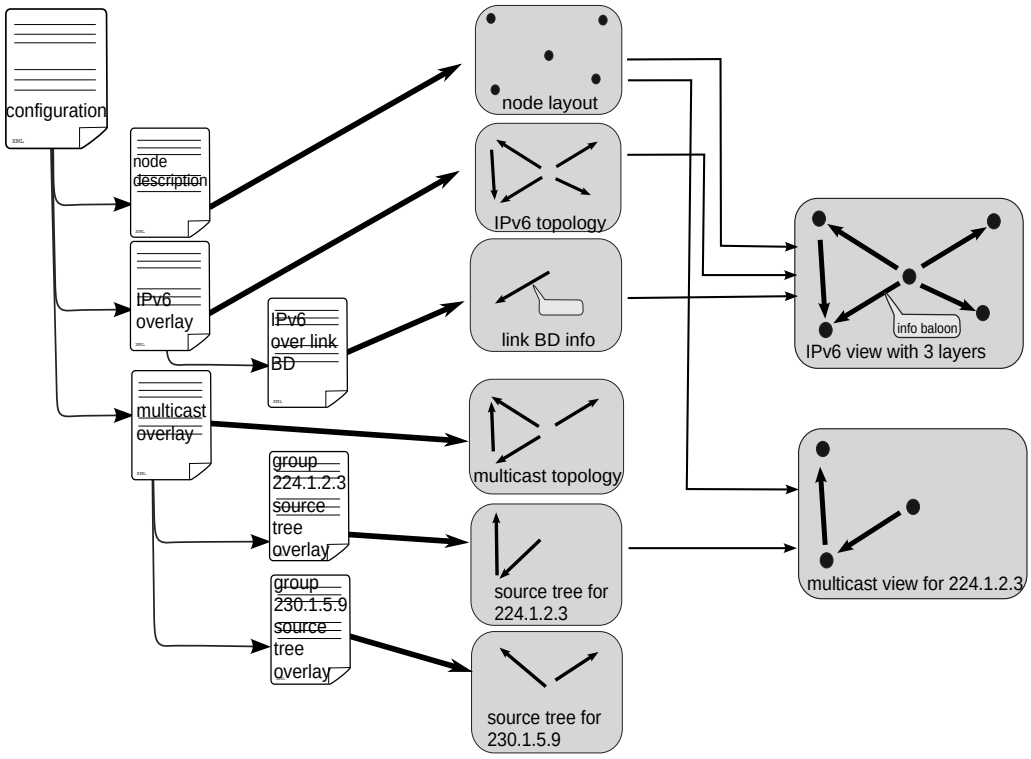

Figure 2: Conceptual arrangement of overlays and dependence to management data documents. Overlays are transposed to produce the final picture.

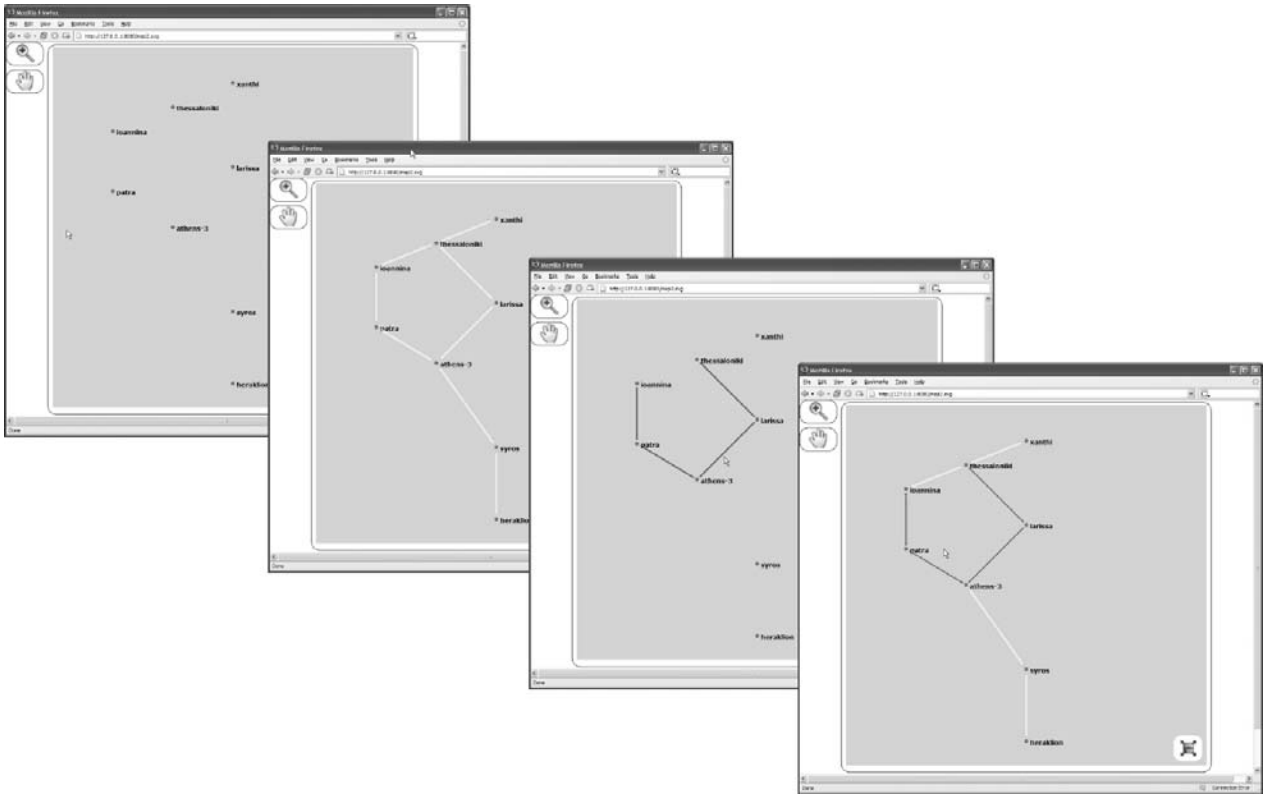

Figure 3: Juxtaposition of 3 different overlays to produce a composite map. The final visualization on the right is produced by combining the other three overlays.

retrieve one or more of these documents and present more information (e.g., which multicast groups are flowing in each direction of the link) to the user.

A conceptual explanation of management data documents and their relation with the presented overlays can be seen in Figure 2. An actual example can be seen in Figure 3.

Management Applets

The main idea behind the selection of the overlay hierarchy is that there are potentially many specialized management tools (management applets) in the server side, and each one of those applets is providing at least one management overlay. The requirements for any applet to plug into the framework are:

The applet must be registered in a configuration document so that clients will be able to know its existence. The registration can be carried out by hand by the administrator when he wishes to plug the new applet to the presentation system.

1. The applet must generate at least one management overlay document, so that a full map overlay can be created. A URL pointing to this document is provided to the clients through the configuration document.
2. The applet may generate as many additional management overlay documents as needed. The URLs for these documents can be typically provided inside other previously loaded management overlay documents.

For example, an applet providing the IPv6 topology of a managed network could be considered. This applet can be written in any suitable programming language and will be invoked periodically inside the NMS. Each time it is executed, it will discover the topology of the IPv6 enabled network and generate a management overlay document that describes it precisely. Management overlay documents can also be created which will be revealing more information about each network link or node. All these documents will be typically placed inside the web document directory of the server to be available for retrieval by the clients.

As long as these applets output XML documents suitable for usage by the clients, the administrator is able to easily create new ones which present new sets of data. The schemas of the XML document categories that were described in the previous list were deliberately crafted to be relatively simple, in order to allow easy manipulation and composition.

Although this is not a strict requirement, it is proposed that all applets have access to a managed entity registry which is essentially a database of all the discovered managed nodes and other useful information about them. When an applet needs to communicate with a specific node, it may need information such as its IP address, its type, its SNMP community string, etc. These values can be supplied separately to each applet by the administrator using configuration files, but it is much more convenient to have a type of common registry so that all applets can access the same set of configuration. Of course, applets will be able to add newly discovered nodes to this registry. Typically, applets that discover various topologies will also add new nodes.

As implied in the previous list, the graph(s) that depict the various network functions (each one supported by an applet in the server side) must exist together inside a single SVG image. Each one of the graphs is organized into an SVG overlay, the visibility of which can be turned on and off at will by the client user. As the visibility of a SVG element effects the delivery of user interaction events to itself, only the overlays that are visible will respond to user actions. The client code will also seek to periodically update only those overlays that are visible to the user. With this strategy, it is assured that only information that the user actually wishes to see gets retrieved by the client application.

Implementation

Overview

The implementation of the system consists of the client application code that does the graph rendering and the server applets that carry out the data collection and XML documents creation.

The client application is based on Javascript, which is the embedded scripting language in all modern browsers. The application code has a double purpose. Its main function is to collect the data from the NMS server and create the SVG graphics, while its secondary assignment is to handle all the user interaction events and maintain the graphics accordingly.

Collection of Data Using XmlHttpRequest

Collection of data is essentially carried out by retrieving the appropriate XML documents from the server. As indicated, retrieval is based on the XMLHttpRequest class. Similarly with many other web applications, the XMLHttpRequest class is used here by issuing a call that registers a handler.

```
conf = new XMLHttpRequest(); //create the object
conf.open("GET", "conf.xml", true); //get the document
conf.onreadystatechange = function() {
 if (conf.readyState == 4) {
 ...
 code handling the document inserted here
 ...
}
```

Example 1: Typical code for retrieving the configuration document.

Example 1 shows typical code for retrieving the configuration document.

This handler will be invoked when the request completes successfully (readyState reaches 4) and its task is to interpret the document that was retrieved. Each of the three schemas of documents (see categories the previous sections) is handled by a specific handler. As they interpret these documents, the handlers will manipulate the graphical content of the SVG page. The application code is programmed to issue XMLHttpRequest.open calls whenever needed, at the application start, periodically or after user interaction.

The programmer has the convenience of treating the XMLHttpRequest object as an XML file or a file of arbitrary format. Using the object as XML is actually preferable, because this circumvents the need to parse the document, a work that would have to be done if it was treated as an arbitrary file.

It is reasonable to expect a lot of complexity to be hidden inside the code that reads the XML structure and creates piecewise the graphical content such as the network graph and other artifacts. Although this is true, there is not too much interdependency between the various steps that are taken to complete this task. So, modification and extension in the future will be easy. Essentially, the whole process of converting a management overlay document into a graphical overlay will be assigned to a method that will be taking the document URI as its argument and will be returning the overlay to be placed inside the main SVG image.

Manipulation of Graphical Content Using the SVG DOM

Manipulation of graphical content is done by using the SVG Document Object Model (DOM). This is also on par with the way other modern web applications operate. The DOM is an object oriented mapping of the structure of a document using classes that map directly onto document elements.

This way, a convenient interface to the content is available to programmatically modify, create or delete whatever aspect of the page. When a new element (for example, a polygon) must be created, an object of appropriate class is instantiated and then attached to a suitable preexisting object inside the document. The SVG specification uses the grouping (G) element heavily to group other real graphical elements together. For example, to create a circle, the programmer would write a function like that in Example 2, and then use it to generate a circle and attach it to another element.

```
document.getElementById("overlay_1a").
  appendChild(createCircle(100, 200, 5));
```

Any graphical element, including grouping elements, can be attached under another G element or even the top SVG element itself, forming hierarchies. The order of rendering is strictly the order of appearance in the DOM, so the way each graphic part is placed relative to the others can be controlled. To visualize each management overlay, elements that belong to this overlay are grouped and the group is placed in the appropriate position inside the document.

From the point where an element has been created, many of its properties can be manipulated afterwards to alter its appearance. To change the color of a filled polygon, the application would access its style as an object property and modify the style property that corresponds to the internal fill color. Almost all conceivable style types are available in the SVG specification, such as fill color and pattern, line stroke and pattern, object opacity, etc. The easily accessible style model eases the programming of visually rich graphs immensely. For example, the code that would alter the opacity of an element would be like:

```
element.style.opacity=1.0;
      //make object fully opaque
```

Underlying Operation.

The parts that modify the SVG graphical content are usually utilized as primitives by the parts of the application that handle all the events, including all user interaction. From the point in time where it is created, an SVG page produces events that trigger the execution of code

```
function createCircle(x, y, r) {
    //create the circle
    var c = document.createElementNS("http://www.w3.org/2000/svg", "circle");
    //set some attributes
    c.setAttribute("cx", x); //x coordinate
    c.setAttribute("cy", y); //y coordinate
    c.setAttribute("r", r); //radius
    c.setAttribute("style", "fill:#f34916");
    //create the grouping element
    var group = document.createElementNS("http://www.w3.org/2000/svg", "g");
    //attach element c
    group.appendChild(c);
    return group; // return the group
}
```

Example 2: Creating a circle.

that the programmer has placed as handler of these events. The event with the biggest influence is of course the onLoad event which basically triggers the creation of the entire page. As mentioned earlier, the document is initially devoid of actual graphics as this is completely handled by the application code. The tasks assigned to the onLoad event are summarized as follows:

1. Loading of the Configuration Document and interpretation of its contents. The configuration document will include a URI to a node topology description document. Additionally, several structured references to URIs of Management Overlay Documents will also be present.
2. Based on the URI of the node topology description document, the nodes are placed on the map using appropriate symbols and their positions will be stored in an internal array for later reference. All the graphical elements that represent nodes, including their labels, are placed in a separate overlay.
3. Each Management Overlay Document reference inside the Configuration Document includes information on how to handle the specific overlay. The administrator will typically want some overlays to be visible from the start, some to be available through a menu and some to be available through specific events, such as hovering or clicking on other parts of the map (nodes, arrows, etc.).

Node description documents and overlay documents contain simplified interaction information that controls the events that will trigger the loading of other overlays.

Overlay documents mostly contain information about arrows, labels and other shapes that their position revolves around the position of nodes. A rule of thumb is that the node description document contains elements that have an absolute position and are (almost) always visible, while overlay documents contain elements that have a relative position that is always calculated according to the position of other elements. The appropriate handlers will create all these artifacts and attach them on their respective overlays using the method that was illustrated earlier.

The application follows a strict strategy of loading overlays only when needed and never in advance. This modular way of operation ensures that the software will consume resources only when there is a clear need. Loading all the overlay documents and caching them in advance would be ill-advised, as in a complex scenario many hundreds of small size overlays may be present. Consider for example a multicast topology network map. The main overlay can be containing the underlying topology of the network (usually derived from the PIM [22] neighbor tables of each node), while there can be a secondary overlay depicting the distribution tree of a multicast group (or, even more, a source/group pair). In many cases there could be hundreds of group addresses present in the multicast routing tables of the network routers, which means that equally as many overlays would have to be available to the user.

Likewise, the reloading strategy of overlays must be carefully chosen. Considering the fact that each XML file that supplies an overlay is being produced on the server side periodically, it would be pointless to reload more frequently than the rate by which the data is refreshed. For that reason, the application will always follow the individual refresh rate of each overlay as it is supplied from the corresponding overlay document. This means that each XML overlay description document has builtin the refresh rate and the applet that produces it at the server side may choose to alter its refresh rate under various circumstances.

The last mechanism of loading overlays is through user interaction on the map itself. For example, hovering or clicking on a network link may activate an additional overlay that is downloaded at that moment and becomes visible. That overlay could incorporate additional information about the traffic that flows through the link and depict it with various means, like a conceptual traffic diagram over time or a report that manifests itself with a pop-up text box near the network link. It is left to the imagination of each implementor of applets, to devise new smart ways of revealing new information in response to user events.

The management overlay document schema allows the possibility to define pointers to the URLs of additional overlays which in turn could define other pointers and so on. A virtually unlimited depth of overlays that potentially activate other overlays can be created that way. Although this capability is not currently implemented, it is certain that very interesting ideas may be derived from the concept. The map could also incorporate overlays that could activate new areas that contain even additional nodes the ones defined in the node description document. An interesting example of this capability would be to have the initial map depict the layer-3 topology of a managed domain, and program the behavior where clicking over a link between two layer-3 nodes would reveal additional layer-2 devices that are positioned between them. For instance, clicking an arrow between two routers reveals the switches that are used to connect them.

Development Platform

The client SVG application is being developed on top of the Mozilla SVG implementation, usually inside the latest stable release of Firefox. The Mozilla implementation is ideal for the development of applications like these, because it already possesses an engine that can understand most of the SVG specification and, additionally, has excellent debugging capabilities built-in. Of these capabilities, the DOM inspector is surely the most important, as it allows to explore the SVG DOM in a tree-like fashion and experiment with changes or observe the effects caused by newly introduced application code.

Initial Management Applets

To test the code and provide real world proofs of concept, a series of management applets have been developed providing various categories of network information.

It is of paramount importance to explore various use cases of our presentation system and test different scenarios of operation regarding our applications.

1. A layer-2 topology discovery management applet has been tested on our university's campus to provide a physical view of the interconnections between all our managed switches. The topology discovery algorithm is based on CDP [23], and the node layout is based on the GraphViz package. This is a good example where an automated layout is most useful because of the large number of nodes that are depicted in the graph. An example can be seen in Figure 5.
2. A multicast topology discovery management applet has been developed based on the experience gained from the development of the multicast weathermap project [10]. The Protocol Independent Multicast neighbor information is used to find the PIM neighbors of each managed nodes and discover the multicast topology of the network. This module is under testing at the Greek Research Network. To give the opportunity to approximate the geographic positions of the routers, layout is handled by hand as their number is relatively small. An example of the visualization produced by this applet can be seen in Figure 4.
3. An IPv6 topology discovery management applet is also available. Using newly available IPv6 MIBs [24], this application can use the prefix information and IPv6 neighbor discovery to layout an accurate depiction of the IPv6 layer-3 topology of a network. As in the case of the multicast applet, testing is been carried out on the Greek Research Network.

Strong Points

Security

From a general perspective, the client-server way of operation of this application coincides with the way of operation of many other software applications with similar requirements. The management data collection from the managed nodes is done solely by the NMS server, which of course increases the security of the system. Additionally, as explained before, the management applets produce XML management documents and place them inside the document directory of a simple HTTP server where they are available for retrieval. The fact that the application runs entirely in the client side and retrieves these static files which have been produced asynchronously, completely isolates the NMS server from the clients, resulting in a very secure scheme.

Scalability

It is expected that the scalability of the system in regard to the number of users that will be able to use it simultaneously, will be excellent. As explained previously, the management documents are simply placed inside an HTTP server. So, the number of clients that use the system

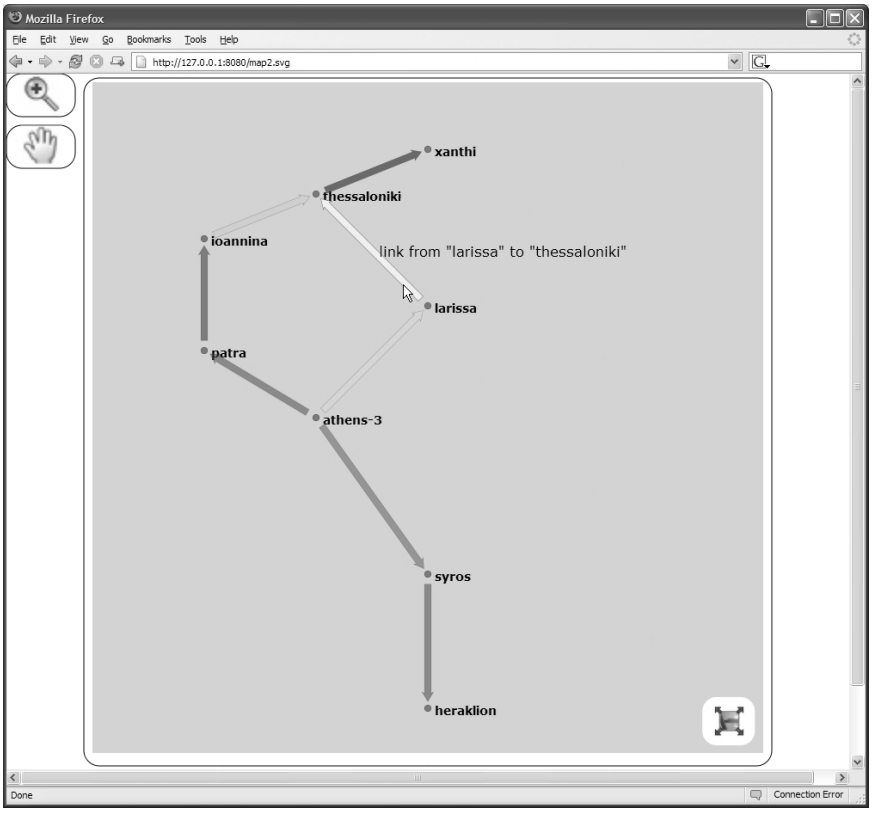

Figure 4: Prototype depicting a multicast topology overlay.

does not affect the operation of the management applets, because these two subsystems are independent from one another. In fact, the HTTP web server could very well be inside a different machine from the NMS server, so even large amounts of HTTP workload would not be placed on the latter. The fact that this architecture can handle a fairly large number of clients, makes it also appealing for creating interfaces that are publicly available. Security is also enhanced as the clients which generally reside on the Internet need not communicate using special protocols, but only simple HTTP operations. Additionally, there are no server side scripting technologies (like CGI scripts, JSP, PHP, etc.) involved, which makes operation of the web server very secure.

It is also expected the the user experience in terms of client speed and responsiveness will be enjoyable. The size of the application code is relatively small and, as mentioned earlier, the application will download new overlay information only when needed.

Related Work

GraphViz

The excellent GraphViz [21] package from Lucent incorporates some very sophisticated algorithms for laying out graphs. Although the tool is in no way associated with the creation of user interfaces or network management, its ability to export its calculated layouts into various formats, including SVG, makes it quite suitable for creating visually impressive network presentations.

As stated before, our tool does in fact use the GraphViz package for the calculations of layouts that contain many nodes. As the GraphViz package does not concern itself with interactivity, it is exceedingly difficult to make it create SVG depictions that incorporate even the simplest forms of user interaction.

Google Maps API

An application that is mostly strange to network management but exhibits notable conceptual similarities with our tool is the Google Maps API [25] which was recently released from Google. Using this API, the Google Maps product can be used as the basis for building other applications. The programmer can create visual artifacts such as arrows, polylines, and placemarks of various sizes and styles and place them on particular areas inside the geographical map.

In principle, this is completely analogous to our solution, where the vertices and edges are defined inside the node topology and overlay description documents. What is even more interesting is the fact that, in the Google Maps API, the user defined elements can be organized into different layers to produce more complex results. Again, this is similar to our own overlays. Combined with the fact that this information is described through XML files and is retrieved with AJAX techniques, the distinct similarities between the two tools can be easily observed.

As an aside, the Google Maps API does not rely on SVG or any other vector based drawing ability on any

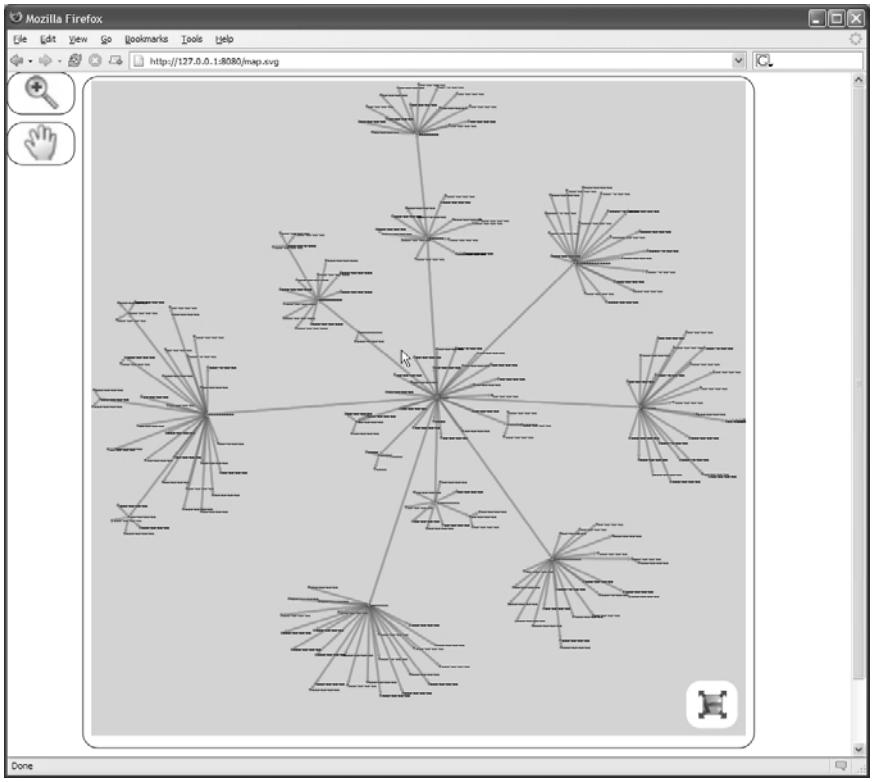

Figure 5: Prototype application depicting a large switched network. User can pan or zoom using the controls on the left.

browser (although it can utilize VML on IE) but uses clever techniques to supplement the lack of those abilities. We believe however that when the usage of SVG becomes more commonplace and is supported natively by the majority of web browsers, it could very well replace all these server aided techniques.

Carto::Net

Carto:Net, a popular web site that provides tutorials on SVG and its usage on cartography, has recently released a tutorial paper entitled ''Dynamic Loading of Vector Geodata for SVG Mapping Applications Using Postgis, PHP and getURL()/XMLHttpRequest()'' [26]. Inside the tutorial, there is the description of a geographical map application that dynamically loads vector data into the map each time the user zooms or pans inside the map area.

While the main focus of the application is of course related to cartography, the fact that data is fetched using AJAX methods and the SVG content is dynamically created on the client side and placed on the map, shows that the way of operation is again here very similar to our architecture.

HP OpenView

HP OpenView [27], one of the oldest and most popular network management framework available, employs the concept of an abstract visualization interface API to provide other network management tools with a platform which can be used to build native (X Windows) or Web based (through Java Applets) user interfaces that concern network management. This API, dubbed ''OpenView Windows,'' allows the programmer to organize network management information into visual submaps that form a hierarchy. Each submap typically incorporates various nodes, unidirectional or bidirectional edges of various colors and styles and other similar constructs. As mentioned earlier, using a graph is the most common approach to depicting a network. A typical network management tool that plugs in to OpenView would have to initiate a series of calls to the API that would create the submaps, populate them with graphs and associate certain actions, such as clicking on a node, with opening other submaps or carrying out other tasks. The Network Node Manager product for instance, uses the OVW API to present its interface to the user. Of course, today there are many more network management tools that use the OVW API as a platform, contributing to its major success.

Certainly, our system has similarities with the OVW API in many regards. The provision of a visualization API to which other tools can incorporate their management information, the usage of graphs to depict networks and the association of user actions to various interactive features are traites that are certainly shared. However, there are still key points that differentiate the overall experience both from the view of the management tool programmer and the end user. For one thing, OVW is a regular API, which means that the programmer has to use a suitable language binding for it, while our system takes a somewhat more liberal approach and imposes no restrictions on the internal architecture or implementation of the management applets. Using the XML management documents that where described earlier, does away with the need to have an API at the language level. On the other hand, the concept of submaps in OVW is superficially similar to our own overlays. It should be noted however, that a closer look reveals that each OVW based tool must create a different set of submaps and that each set can share no information with other groups. This in essence means that all the tools that plug in to an OpenView OVW installation not only work separately but also appear to be separate from the user perspective. In our approach, we sacrifice some of this compartmentalization and have all the overlays share the same node topology (which is described by the node topology description document) and draw on the same area.

Future Work

SOAP

Once the development of this system is finished, an interesting area to explore will be the inclusion of functionality that provides the ability of two way communication between the client applications and the server. Until now, the architecture clearly dictates that information flows only from the server to the clients. Adding the capability to let the client communicate information back to the server would enable possibilities such as saving various client side settings and retrieving them at the next session, executing various management functions on the management nodes, or even communicating operation instructions to specific server applets. The idea of using HTTP POST operations or even SOAP [28] calls seems like the right way to go on that regard. Modern browsers will most likely support SOAP operations (Mozilla already does in a limited fashion), so this is a very interesting possibility.

Management Document Hierarchy Simplification

The distinction between different types of management applet generated XML documents is also a point of extended discussion. It is quite possible that at least node descriptions and overlays may merge in a common schema which can be easily handled by a unified handler on the client side. The real problem is having different applets cooperate efficiently on the server side for the creation of these super-documents. It could be argued that merging those two schemata completely, could require further cooperation between the applets that produce them, thus making their development more cumbersome. This is definitely undesired, so great caution must be exercised in that area.

Currently the client application uses a very simplified model to handle various user interactions as they are declared inside the overlay description documents. As a generalization, a full meta-language that defines actions and events could be developed that could lead to substantially richer and more complex presentations.

The initial thoughts about this meta-language is that it will be surely mapped to the overlay document schema and that it will be similar in its feel to a functional programming language.

SVG Animation Capabilities

On the graphics plane, the usage of animation capabilities that are defined in the SVG specification may lead to an even more impressive presentation of network management data. Unfortunately, some SVG implementations do not have these capabilities yet, a fact that makes development troublesome.

Conclusion

Our experience indicates that the level of complexity and diversity to which the modern network management landscape has come to, often places a difficult task on the network administrators. In addition, these people often have to deal with complex management systems and, even worse, cumbersome and non-intuitive interfaces. The solution that is proposed on this paper tries to borrow ideas from the web by using the browser as a rich client and utilizing modern techniques like SVG and AJAX to provide attractive graphics. At the same time, the difficulty of extending the system is exceptionally low. Indeed, the network administrators can implement their management applet using any method or programming language they desire, and install it easily. Even better, the Javascript application which renders the graphics on the client side, will rarely require modifications. Combined with the fact that the XML schemata that govern the communication between the server and the clients were deliberately engineered with simplicity in mind, it is easy to reach the conclusion that the barrier to extend this system has been kept low. The application delivery method that is presented in this paper exhibits similarities with the classical approach of using Java applets to implement a rich client inside the browser. However, the protocol that is used to transport information is simple HTTP and the content is more presentation driven than management driven. Arguably, this strategy can lead to more compact communication volumes. The content is also encapsulated using XML, which does away with the need to have a special protocol and parser. Lastly, extensibility is benefited because of the usage of standards and openness of the architecture.

The demonstrated method of using AJAX to produce an interactive application that downloads its parts on demand, along with the usage of SVG to create vector based graphics that change during execution and respond to user actions, outlines a philosophy for the development of future web based applications. Furthermore, the method of overlay loading which has been outlined in this paper, advertises the strategy of loading graphics and interaction data on demand as the application is used. So, it is our hope that with our case we have inspired new ideas and innovative approaches on other problem areas besides network management.

Author Biographies

Athanasios Douitsis, born in 1976, graduated from the Department of Electrical and Computer Engineering of the National Technical University of Athens in 2000 and is currently a Ph.D. candidate at the Network Management and Optimal Design Laboratory at NTUA. He has been working for the NTUA Network Operations Center since 2000, involved in the administration of the NTUA campus network, the Greek Research Network (GRNET), the Greek School Network and the Greek Student Network. He has experience in Network Management, Monitoring and Measurements, Multicast, IPv6, VPNs and system administration.

Dr. Dimitrios Kalogeras was born in Athens in 1967. He graduated from the Department of Electrical and Computer Engineering of the National Technical University of Athens (NTUA) in 1991 and in 1996 he acquired the Doctoral diploma from the same department. Dr Kalogeras has participated in numerous research programs of the EC and the General Secretariat of Research and Technology in Greece. He has pioneered in the design and development of the NTUA and the GRNET data networks and is a member of the technical and scientific committee of the Greek School Network. Dr Kalogeras is a consultant on issues on networking and video signal processing. He is also the author and coauthor of publications in international magazines and proceedings of numerous conferences. From 2000 to 2002 he has served as a member of the Terena Technical Committee.

References

[1] *Scalable Vector Graphics specification*, http://www.w3.org/Graphics/SVG/ .

[2] *Ajax: A New Approach to Web Applications*, http://adaptivpath.com/publications/essays/archives/000385.php .

[3] *The X Window System*, http://www.x.org/ .

[4] *VNC*, http://www.realvnc.com/ .

[5] *Citrix ICA protocol*, http://www.citrix.com .

[6] *Understanding the Remote Desktop Protocol (RDP)*, http://support.microsoft.com/kb/186607 .

[7] *SunRay Technology*, http://sun.com/sunray .

[8] *Java Applets*, http://java.sun.com/applets/ .

[9] *GD graphics library*, http://www.boutell.com/gd/ .

[10] *The Multicast Weathermap*, http://netmon.grnet.gr/multicast-map.shtml , http://www.terena.nl/events/archive/tnc2004/programme/presentations/show.php?pres_id=47 .

[11] *Vector Markup Language*, http://www.w3.org/TR/NOTE-VML.html .

[12] *Adobe Flash*, http://www.adobe.com/support/documentation/en/flash/ .

[13] *Adobe SVG*, http://www.adobe.com/svg/ .

[14] *Opera Web Browser SVG implementation*, http://www.opera.com/products/desktop/svg/ .

[15] *Mozilla SVG project*, http://www.mozilla.org/projects/svg/ .

[16] *Batik SVG toolkit*, http://xmlgraphics.apache.org/batik/ .

[17] *Java Web Start*, http://java.sun.com/products/javawebstart/ .

[18] *The Simple Network Management Protocol*, http://www.ietf.org/rfc/rfc1157.txt .

[19] *Javascript*, http://www.mozilla.org/js/ .

[20] *The XmlHttpRequest object*, http://xulplanet.com/references/objref/XMLHttpRequest.html .

[21] *Graphviz graph visualization package*, http://www.graphviz.org/ .

[22] *Protocol Independent Multicast*, http://www.ietf.org/html.charters/pim-charter.html .

[23] *Cisco Discovery Protocol*, http://www.cisco.com/en/US/tech/tk648/tk362/tk100/tsd_technology_support_sub-protocol_home.html .

[24] *IPv6 IETF charter*, http://www.ietf.org/html.charters/ipv6-charter.html .

[25] *The Google Maps API*, http://www.google.com/apis/maps/ .

[26] *Dynamic Loading of Vector Geodata for SVG Mapping Applications Using Postgis, PHP and getURL()/XMLHttpRequest()*, http://www.carto.net/papers/svg/postgis_geturl_xmlhttprequest/ .

[27] *HP Network Node Manager*, http://www.openview.hp.com/products/nnm/ .

[28] *Simple Object Access Protocol*, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ .