A Client-Server Architecture for State-Dependent Dynamic Visualizations on the Web

Daniel Coffman, Danny Soroker, Chandra Narayanaswami
IBM T.J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
{coffmand, soroker, chandras}@us.ibm.com

Aaron Zinman§
MIT Media Lab
75 Amherst St., Cambridge, MA 02142
azinman@media.mit.edu

ABSTRACT
As sophisticated enterprise applications move to the Web, some advanced user experiences become difficult to migrate due to prohibitively high computation, memory, and bandwidth requirements. State-dependent visualizations of large-scale data sets are particularly difficult since a change in the client's context necessitates a change in the displayed results. This paper describes a Web architecture where clients are served a session-specific image of the data, with this image divided into tiles dynamically generated by the server. This set of tiles is supplemented with a corpus of metadata describing the immediate vicinity of interest; additional metadata is delivered as needed in a progressive fashion in support and anticipation of the user's actions. We discuss how the design of this architecture was motivated by the goal of delivering a highly responsive user experience. As an example of a complete application built upon this architecture, we present OrgMaps, an interactive system for navigating hierarchical data, enabling fluid, low-latency navigation of trees of hundreds of thousands of nodes on standard Web browsers using only HTML and JavaScript.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures – *Patterns (e.g., client/server, pipeline, blackboard)*.

H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces – *Web-based interaction*.

General Terms
Performance, Design, Human Factors.

Keywords
Rich Internet Applications.

1. INTRODUCTION
Enterprise applications are moving to the Web for a variety of reasons, including ease of deployment, manageability and consistency and security of enterprise data. Achieving the performance to which users have become accustomed on traditional applications requires new frameworks and methodologies in the Web client-server model. This challenge is further exacerbated when large quantities of data have to be presented visually and altered dynamically as the user's context changes. In this paper we present one such endeavor for the display and navigation of layered hierarchical data. Our client-server model performs several orders of magnitude faster compared to a direct port of a traditional model, in the context of applications that support visualization and interactive navigation.

The concrete application we discuss here is called *OrgMaps*, whose goal is to visually map hierarchical organizations and also reflect the superposition of additional data (visual mashups). OrgMaps permits users to navigate smoothly the structural *neighborhoods* of individuals within the organization – their department, reporting chain and so on – through zoom and pan operations. The visualization also functions as a substrate for the overlay of additional information.

One of the fundamental challenges in building OrgMaps was scalability: making it perform well for large organizations with hundreds of thousands of individuals. To address this challenge we employ a tiling methodology where client-specific tiles are rendered server-side on demand. Central to this design is the dynamic construction of small view-dependent tiles in image space depicting the data, and the delivery of those tiles with related artifacts describing the user's current region of interest. The associated artifacts may be quickly and easily updated based on the user's interactions, leading in large part to the quick responsiveness of the application's interface. We believe that the techniques presented here, manifested originally for OrgMaps, are applicable to the design of a large class of Web applications.

2. VISUAL DESIGN
2.1 Requirements and Goals
We built an earlier Java-based prototype in order to rapidly explore the design space for visual mapping of organizations. Our choice of visual design builds upon one of the simplest hierarchical layouts, the *icicle plot* [1]. Icicle plots place parents directly above their children, keeping edges implicit rather than explicit. In this way, the plot can be called space-filling. Each node is represented by a rectangle whose width is the sum of the widths of its children. All nodes have the same height, and all leaf nodes have the same width (for a given zoom level).

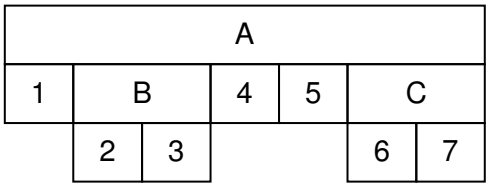

Figure 1: Sample icicle plot

§ Work performed while at IBM TJ Watson Research Center.

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

Figure 1 shows an icicle plot for a small organization with 10 people: 3 managers (nodes A-C) and 7 non-managers (nodes 1-7). The reporting structure is very easy to grasp by glimpsing at the figure (e.g., 3 reports to B, and B reports to A). This ability to follow parentage vertically is a primary reason we chose icicle plots over alternative layouts.

2.1.1 Base Visualization

Figure 2 shows a screen shot of our interactive implementation of an icicle plot for a fictitious organization of 150 people.

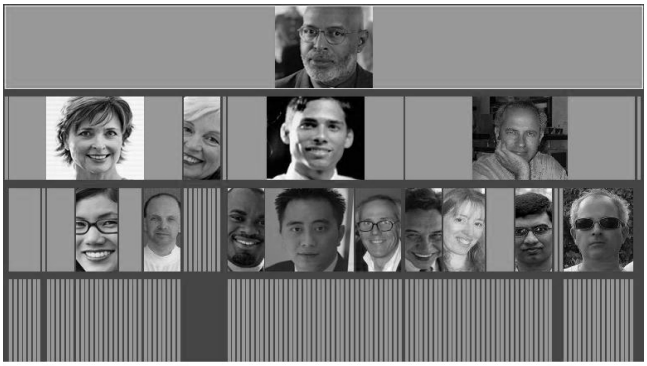

Figure 2: Global view of an organization

OrgMaps uses faces as a central aspect of its visualization. It builds upon human ability to quickly recognize faces and thus help form a visual memory of the organizational structure that a user builds up over time. As the entire organization is visible, leaf nodes become very thin. Only nodes that are wide enough (beyond a threshold we set) show the face of the person they represent (in this case, 14 of the 150 faces are visible). However, by instrumenting OrgMaps as zoomable and pannable, we can investigate all branches and individuals in a method similar to starting with a map of the US, zooming in to a city, and then panning to locate its various neighborhoods. Via a user interface gesture we can zoom in on a person so that they become the focus of the plot, as shown in Figure 3. Note that, even when zoomed, faces of the complete management chain are kept fully visible for improved context and navigation.

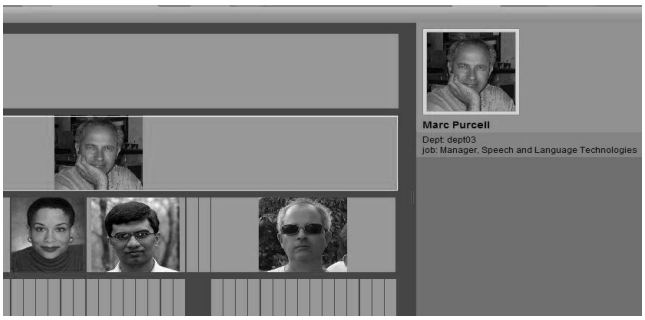

Figure 3: Zoomed-in view of a department

This figure also shows the details panel to the right of the plot, in which information about the selected person is presented. Both the faces and the displayed information are obtained from a centralized corporate directory.

2.2 Interactions

OrgMaps supports highly responsive interactions, among them selection, zoom/pan and search.

Selection: an individual is selected through keyboard or mouse action; the individual is highlighted and specific detailed information is displayed in a separate area.

Zoom, pan: the map may be dragged horizontally to pan it; the zoom level may be adjusted to reveal more or less information about the neighborhood of an individual.

Search: People can be searched for either globally or contextually.

3. SCALABLE WEB SOLUTION

3.1 System Architecture and Overview

The architecture is designed for maximal efficiency. The data are maintained in a database; the data are fetched from the database when the server is initialized and thereafter maintained by the server in its memory. The data are processed into a set of models ranging from general to specific, as shown in Figure 4. The models are designed to minimize memory footprint while maximizing sharing of data among users. The single largest model is the Abstract Data Model. It contains as much of the data from the database as practicable, and is shared among all clients. The Abstract Data Model is reconstructed periodically as changes to the organizational structure are reported. The architecture allows the database to be refreshed independently of the of the in-memory models.

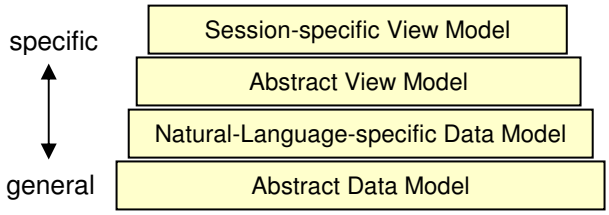

Figure 4: Server-side layered models

The client browser and the application establish a session, during which time the client offers the server metadata describing itself: its screen size, resolution, and the user's preferred language. The server inspects its internal structures to determine if an appropriate *Natural-Language-specific Data* Model has already been constructed, and if not creates and stores it for future lookups. This Natural-Language-specific Data Model is not a language-specific copy of the Abstract Data Model, but is a filter placed before it to replace tokens with translated phrases upon access. Natural language-specific data models are shared among all relevant clients. This model is of very modest size, requiring only about 0.5% of the memory used by the Abstract Data Model.

The user initiates the process of viewing a hierarchy, or *tree*, by specifying the identity of the individual at the root of the tree. This choice is sent to the server which in turn creates an *Abstract View Model* of this tree, representing a view of the tree suitable for arbitrarily fine resolution and arbitrarily large screen size of the client. This model contains only information on the positions of individual nodes within the view and may depend on the particular natural language. It also is of modest size. The server temporarily stores this model so that it may be shared among all clients viewing a tree from the same root in a Least Recently Used (LRU) cache.

The server uses the information on resolution and screen size provided during session creation to construct a *Session Specific View Model* for the client. This is the only model unique to a

particular client and session. It incorporates knowledge of the resolution of the client device, client identity, and other client and session-specific data. The server derives the images it sends to the client directly from this model. Further, the server computes and maintains a set of coordinate transformations from the space of the client device to that of the session specific view model so that the user's actions, such as the moving of the pointing device, may be mapped efficiently to the corresponding element in the Session Specific View Model.

Additional metadata may be associated with an individual in the data set's model, such as a picture or label. Such metadata is displayed only if the individual's node as represented in the view model is large enough so that it would be visible. The server adds in this metadata for nodes larger than a threshold value, thereby reducing unnecessary data transmitted.

The server takes the zoom-level into account in several places. When the zoom-level is sufficiently high, the view model may represent an image many times larger than the available area on the client device. Delivering a single image of the view model in this case would be inefficient and unnecessary. The server prepares, rather, a set of *tiles*, one or more for each level of the hierarchy. Suppose that the user is currently interested in a particular region of the view model. The set of tiles prepared for this region would comprise tiles covering the region and also the regions immediately to the left and right of the region of interest. Tiles beyond the perimeter fences are ignored.

Delivery of the tiles proceeds in phases; initially, only the bounding box of the tile need be delivered to the client, along with a unique tile key. The client uses this key when constructing the URL for fetching the image contained in the tile. Tiles extending beyond the left and right perimeter fences are truncated by the client at the fences before such a request is placed. The server takes this truncation into account while drawing the tile's image. Individuals in the organization each belong to a single tile, as splitting an individual across tiles could lead to a highly disruptive flicker when the tiles are displayed. In addition to the images prepared for the client, the server prepares a limited set of descriptors, delimiting various regions of the images.

It is the responsibility of the client application to assemble the set of tiles and descriptors it receives into a coherent presentation for the user. Further, it maintains a series of linked-lists containing the descriptors as they arrive from the server.

Given the very large available address space of our server, we choose to maintain all of the objects described above in the server's memory. This naturally leads to the best performance by the server at the cost of a substantial memory footprint. The single largest object is the Abstract Data Model. For a dataset of twenty-four thousand individuals, this requires about 34 megabytes. The memory required for any individual user is much smaller, being initially about 300 kilobytes and increasing slowly in size to about 10 megabytes as the user interacts with the system.

3.2 User Interaction

The tiles must be adjusted after the completion of a user interaction. After a pan operation, the positions of faces must be recalculated so that they remain completely visible, if possible. In addition, new tiles must be computed and delivered corresponding to regions still hidden, but likely soon to come into view. After a zoom operation, the entire complement of tiles needs to be recalculated.

3.3 Implementation Setup Details

We make use only of dynamic HTML, Asynchronous JavaScript and XML (AJAX), and HTTP servlets composed in the Java language. For purposes of this paper, we wished to investigate the limits of HTML and AJAX, determining by how much we could constrain their resource usage, in the hope of extending this work to mobile devices, with very limited memory available and without additional runtimes beyond the web browser. For similar reasons we avoid reliance on rendering technologies such as Adobe Flash or Microsoft SilverLight.

4. EVALUATION

4.1 Experience and Feedback

Early versions of OrgMaps have been demonstrated both within IBM and at the Lotusphere conference in 2008. As the corporate directory is one of the most heavily used enterprise applications at IBM, there was clear interest in OrgMaps' ability to provide easily-navigable views and data aggregation. People from other types of organizations, such as government and education, also saw clear use cases for the hierarchical view. The desire of people to easily deploy OrgMaps for their organization was an important factor in leading us to pursue a Web-based implementation.

4.2 Performance

One of the most important considerations for the user of an interactive application is the amount of time required before the application is loaded and ready for operation. Another factor is the responsiveness of user interactions. In our initial prototype implementations of OrgMaps we used an architecture whereby a complete description of the organization was delivered to the client browser. The browser was then able to perform all actions required by the user, the server acting only to provide metadata pertaining to a selected node, as needed. The architecture performed well for small organizations, but was unsuccessful for large ones. The performance of a client system using only HTML and JavaScript is illustrated in Figure 5.

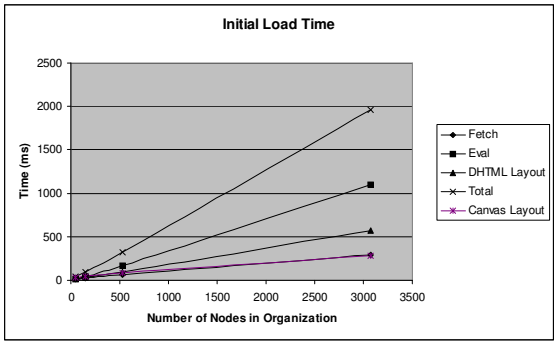

Figure 5: Load Times for a client centric Architecture

The figure shows the times to fetch the organization into the browser, the time to *eval* or transform this into JavaScript objects, and the time to *layout* or create objects in the browser's DOM to render the organization visible. An organization of only 3000 nodes requires almost two seconds to be usable in such a scheme. It did not prove feasible to view an organization of 20,000 nodes; the time to deliver the organization alone rose to over two minutes. We tried replacing the use of DOM objects in the browser with a Canvas as implemented in Firefox and Safari. This yielded a slight improvement in performance in that the layout

time was reduced, but the dominant *eval* time was naturally unchanged. This is also illustrated in Figure 5.

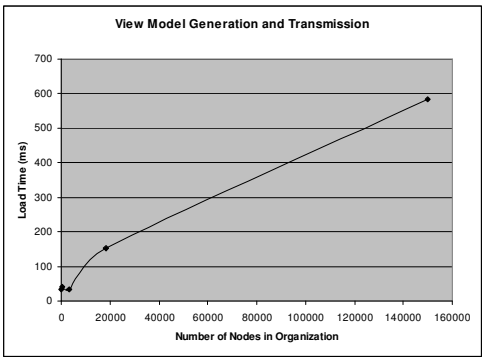

Figure 6: Creation and Load Times for a scalable architecture

Clearly, a different approach was needed to rectify these shortcomings for very large organizations. We chose to partition the data into a set of models maintained in the server, and a much smaller set delivered to the client, as detailed in Section 3.1. The performance is remarkably improved. For example, consider an untiled view; here all of the nodes in the organization may be rendered in a single image lying within the viewport. The server creates the Session Specific View Model, and renders it in an off-screen buffer. It only needs to deliver to the client the descriptors of a few individuals in the vicinity of the selected individual and a *view key*, used subsequently by the client to fetch this image. The eval time has been reduced to an insignificant 3 ms. The time needed to fetch an organization is shown in Figure 6. It is possible to fetch organizations of as many as 150000 nodes with an acceptable response time of less than one second. Note that this time includes the layout time of Figure 5 since the layout and drawing of the image is performed by the server before it returns the set of descriptors to the client.

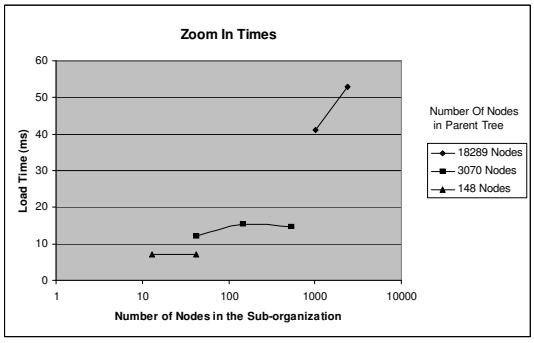

Figure 7: Time to Create and Load *Tiled* View

Consider next a *tiled* view, typically created through a process of zooming in. Further, assume that such a view was created by zooming-in within an untiled view. The times to create and load the tiled view are shown in Figure 7 for untiled views of three different sizes. The times depend only weakly on the number of nodes in the tiled view and depend most strongly on the number of individuals represented by the untiled view. This is to be expected, as the tiled view is derived from the Session Specific View Model, which contains the entire contents of the untiled view. The choice of a sub-organization of a particular size only necessitates the location of a particular region within the Session Specific View Model.

We paid particular attention to the size of the various models on the server. It proved possible to limit the Abstract Data Model for an organization of twenty-four thousand individuals to 34 megabytes, the size being linear in the number of individuals. For an organization of similar size, the Natural Language Specific Data Model requires roughly 110 kilobytes. The view models are only created when the user initiates a request. The initial, untiled, view of this organization requires 330 kilobytes, while a tiled view requires an additional six megabytes.

5. FUTURE WORK

We plan to extend our work in two significant ways, namely building out several additional features and capabilities into OrgMaps, and applying these techniques to other applications with large datasets. Examples might be applications such as representations of product catalogs and educational, governmental, and professional organizations. As mentioned in the beginning of this paper, one of the motivators for moving to the Web is composition of services. We plan to integrate the OrgMaps service into other applications such as mail, calendar and meetings, where recipients and attendees can be highlighted to generate an OrgMap view. Further, we plan to integrate OrgMaps with collaborative facilities such as instant messaging whereby a chat could be initiated when the user clicks on a node on the OrgMap. Further, the instant messaging client will be able to indicate the availability of other individuals through a visual artifact on the OrgMap itself.

6. RELATED WORK

Our work is principally differentiated from previous work in that, 1) it uses a basic Web client, 2) its fluid, low-latency response necessitates a tiling of the visualization with look-ahead, 3) when look-ahead tiles become the central focus in the client viewport, these tiles reflect a constrained-view that is not equivalent to the previously focused tiles on a larger viewport, and 4) we dynamically classify the view-specific tiles to reduce unnecessary communications. For detailed discussion of related work, as well as further elaboration on all sections, please refer to [2].

7. CONCLUSION

In this paper we have presented the OrgMaps system for interactive mapping of hierarchical organizations. The scalable Web architecture we have devised enables OrgMaps to perform well even for organizations with hundreds of thousands of people. We believe that the architecture and methodology described here are broadly applicable to Web-delivered visualization-intensive enterprise applications.

8. REFERENCES

[1] Kruskal, J.B, and Landwehr, J.M. 1983. “Icicle Plots: Better Displays for Hierarchical Clustering”. The American Statistician, vol 37, no 2. pp. 162-168.

[2] Coffman, D et al. “A Client-Server Architecture for State-Dependent Dynamic Visualizations on the Web”. IBM Research Report, RC 24946.