# 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[1]
MIT Media Lab
20 Ames Street, 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. Large monolithic applications are also being decomposed into Web-based services so they can easily be combined with other services. Transitioning an enterprise application into a responsive and scalable Web application is not straightforward when it demands non-trivial amounts of computation, memory, bandwidth, and other scarce resources. This challenge is further exacerbated when large quantities of data have to be presented visually and altered dynamically as the user's context changes.

[1] This work was done while at IBM TJ Watson Research Center

Achieving the performance to which users have become accustomed on traditional applications requires new frameworks and methodologies in the Web client-server model. In this paper we present one such endeavor for the display and navigation of layered hierarchical data. We motivate changes to a traditional application architecture, in which the application runs completely on the client's machine with any server acting only to provision this application with data; we seek to achieve high-performance inside any standard Web browser using common technologies. Our client-server model performs several order**s** of magnitude faster compared to a direct port of a traditional model, without sacrificing dynamic behavior on the client.

An important consideration in designing high-performance Web applications is the partition of labor between client and server, both in processing and transmission of data. Towards one end of the spectrum, some enterprise applications, such as those used in the interactive exploration of large-scale data sets, push large amounts of data to the client to distribute the computation away from central data-centric servers. Such an approach is not yet plausible in the modern Web browser; some of the strongest limitations include single threaded virtual machines, too much heterogeneity in (the lackluster) performance and capability of native graphics functions, overhead with AJAX-based network I/O, and a DOM model optimized for incremental flow of text rather than specialized drawing technologies and atomic animation. In other cases, even if the browser had access to onboard graphics accelerators and bandwidth was plentiful, enterprises may not wish to send the raw application data to each user's browser for security reasons, preferring instead to send intermediate forms of processed data that are less revealing. These issues and more are particularly problematic when trying to support even more constrained clients such as mobile phones using the same HTML and JavaScript as for the desktop.

We tackle these performance issues 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 with the superposition of additional data (visual mashups). OrgMaps, shown in various figures in Section 2, permits users smoothly to navigate the structural *neighborhoods* of individuals within the organization – their department, reporting chain and so on – through zoom and pan operations. Visually overlaying data on the organization structure is often highly informative, e.g., seeing the distribution of children and teachers in an elementary school who have had the flu this season. OrgMaps belongs to a class of applications we call *Enterprise Mashup Substrates*, in which enterprise data is presented visually in a common form to act as a generic substrate for overlaying associated information.

One of the fundamental challenges in building OrgMaps was scalability: making it perform well for large organizations with hundreds of thousands of individuals. Our initial approach of sending the organizational data to the browser for rendering did not perform well for organizations larger than a few hundred members for reasons cited above, which are discussed in more detail in section 4.2. We then considered approaches used in graphics rendering for thin clients where texture maps (e.g., games) and pre-rendered tiles (e.g., Google Maps) are often used instead of more exact object representations to speed up rendering. A major roadblock in our visualization was the inability to use static pre-rendered image tiles. Static tiles prevent arbitrary zoom levels, do not keep up with dynamic nature of the input data being concurrently modified, cannot support fine-grained access controls, and cannot flexibly support view-specific morphology. Instead, we looked to a tiling methodology where user-specific tiles are rendered server-side on demand.

The key contribution of this paper is a novel methodology for building scalable Web applications, which present maps of structured data in a Zoomable User Interface (ZUI) similar to Google Maps or Microsoft's Seadragon. 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. In particular, they are very suitable for the implementation of semantic 2D ZUIs.

The organization of this paper is as follows. We first present the design of OrgMaps as a way to show hierarchies in a space-filling visual map that is amenable to fluid zoom and pan interactions. Next, we demonstrate the unique features of OrgMaps by describing an earlier Java prototype that used traditional enterprise client-server methodologies, and discuss its limitations. We then discuss the scalable and highly responsive Web-based solution: its architectural design, pertinent implementation details, and its performance evaluation. Finally, we present related work, discuss several in-depth aspects and future plans, and conclude.

## 2. VISUAL DESIGN & JAVA PROTOTYPE

### 2.1 Requirements and Goals

We initially built a series of Java-based prototypes in order to rapidly explore the design space for visual mapping of organizations. Our design choices were guided by the following primary constraints:

1. The ability to gain global impressions while exploring local details of an organizational hierarchy, transitioning between the two ends smoothly and rapidly.
2. The ability to easily associate, overlay, and visualize various forms of data contextually in the organizational hierarchy.

These requirements led us to the following concrete interaction design principles.

1. Simple layout of the hierarchy.
2. Fluid semantic zoom/pan interaction.
3. Generic mechanisms for associating data.
4. Simple metaphors for visualizing overlaid data.

Our eventual choice of visual design builds upon one of the simplest hierarchical layouts, the *icicle plot* [7]. 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).

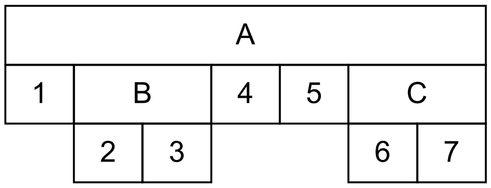


**Figure 1: Sample icicle plot**

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 organizations, OrgMaps, for a fictitious organization of 150 people called *renovations.com.*

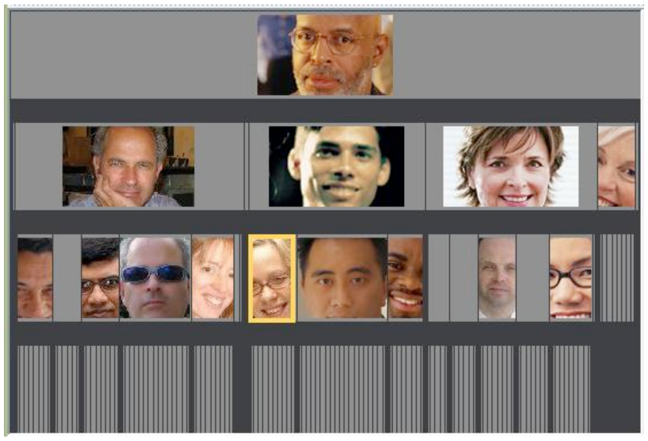


**Figure 2: Global view of an organization**

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 a ZUI, 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.

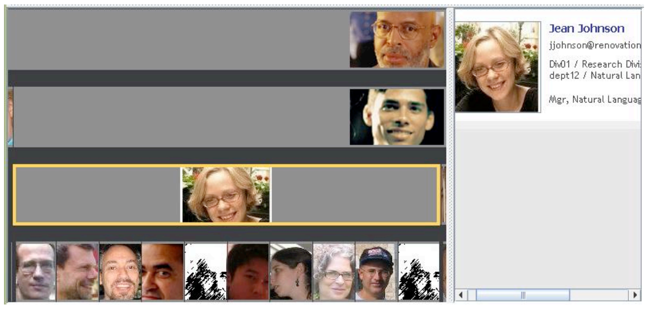
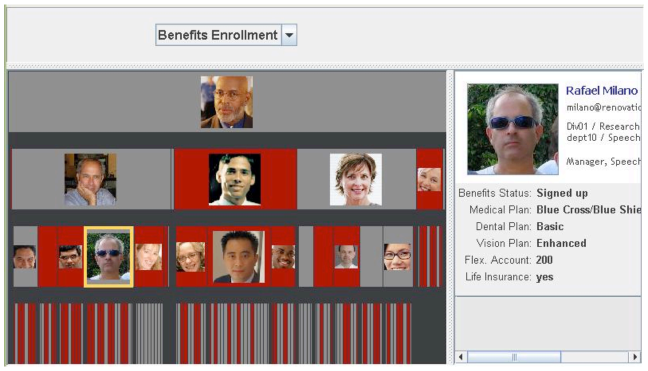

Jean Johnson
jjohnson@renovation
Div01 / Research Divi
dept12 / Natural Lan
Mgr, Natural Languag

**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.

### 2.1.2 *Visualizing Mashups*

OrgMaps supports Boolean and scalar variables for visualizing node attributes. The actual data source may show more data in the details view per person. Figure 4 depicts a sample Boolean mashup, showing, for each person, whether they have enrolled for benefits this year.

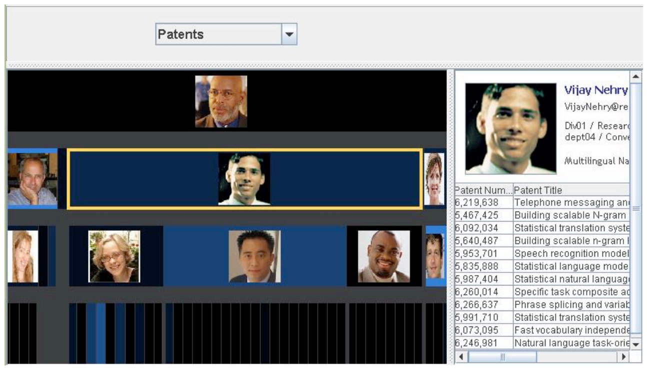


Rafael Milano
milano@renovatio
Div01 / Research
dept10 / Speech
Manager, Speech

| | |
|---|---|
| Benefits Status: | **Signed up** |
| Medical Plan: | **Blue Cross/Blue Shie** |
| Dental Plan: | **Basic** |
| Vision Plan: | **Enhanced** |
| Flex. Account: | **200** |
| Life Insurance: | **yes** |

**Figure 4: Boolean mashup (faces are shrunk)**

This mashup was customized to highlight "trouble" regions, so red cells highlight individuals who have not signed up for benefits. Such people are easily found through gestalt. Note that here faces were shrunk, so that the mashup values would be easier to see.

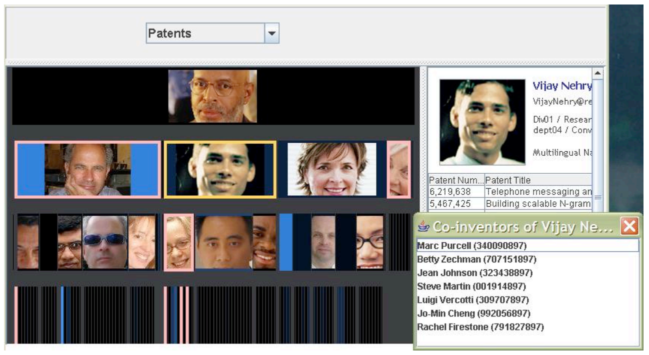


Vijay Nehry
VijayNehry@re
Div01 / Researc
dept04 / Conve
Multilingual Na

| Patent Num... | Patent Title |
|---|---|
| 6,219,638 | Telephone messaging an |
| 5,467,425 | Building scalable N-gram |
| 6,092,034 | Statistical translation syste |
| 5,640,487 | Building scalable n-gram l |
| 5,953,701 | Speech recognition model |
| 5,835,888 | Statistical language mode |
| 5,987,404 | Statistical natural languag |
| 6,260,014 | Specific task composite ac |
| 6,266,637 | Phrase splicing and variab |
| 5,991,710 | Statistical translation syste |
| 6,073,095 | Fast vocabulary independe |
| 6,246,981 | Natural language task-orie |

**Figure 5: Scalar mashup**

Figure 5 shows a scalar mashup of the organization's issued patents (based on real patent data from the USPTO Web site). This mashup uses color intensity to reflect the number of patents, where a brighter blue background reflects more patents. All patents of the selected individual are displayed in the details view.



Vijay Nehry
VijayNehry@re
Div01 / Resear
dept04 / Conv
Multilingual Na

| Patent Num... | Patent Title |
|---|---|
| 6,219,638 | Telephone messaging an |
| 5,467,425 | Building scalable N-gram |

Co-inventors of Vijay Ne...
Marc Purcell (340090897)
Betty Zechman (707151897)
Jean Johnson (323438897)
Steve Martin (001914897)
Luigi Vercotti (309707897)
Jo-Min Cheng (992056897)
Rachel Firestone (791827897)

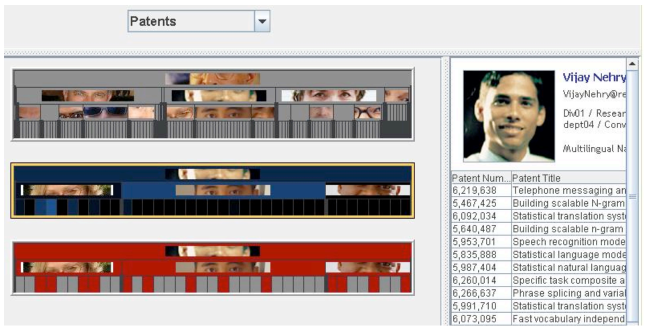
**Figure 6: Interpersonal connections**

### 2.1.3 *Visualizing Cross-Organizational Connections*

Beyond the departmental structure, we often wish to see other connections within an enterprise. As an example, Figure 6 shows patent collaborators on the map by highlighting a selected person in one color (yellow), and the co-inventors in another (pink). The list of connected people is displayed in the results dialog (on the bottom right). Selecting a name on this list focuses the map on that person.

### 2.1.4 *OrgMaplets and Overview View*

The screen shots thus far show detailed views of a single organization. Our system enables instantiation of multiple organizational snippets, called OrgMaplets. Figure 7 features two OrgMaplets, created from the larger map on top.



Vijay Nehry
VijayNehry@re
Div01 / Resear
dept04 / Conv
Multilingual Na

| Patent Num... | Patent Title |
|---|---|
| 6,219,638 | Telephone messaging an |
| 5,467,425 | Building scalable N-gram |
| 6,092,034 | Statistical translation syst |
| 5,640,487 | Building scalable n-gram |
| 5,953,701 | Speech recognition mode |
| 5,835,888 | Statistical language mode |
| 5,987,404 | Statistical natural languag |
| 6,260,014 | Specific task composite a |
| 6,266,637 | Phrase splicing and varial |
| 5,991,710 | Statistical translation syst |
| 6,073,095 | Fast vocabulary independ |

**Figure 7: Overview view**

In this example, the OrgMaplets are created to expand select portions of the organization. Aside from structural zoom, each OrgMaplet can reflect a different mashup as specified by the user, thus simultaneously presenting a multiplicity of views into the organization.

## 2.2 Interactions

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

*Selection:* Moving the mouse sets the hovered person as selected – they are highlighted on the map and their data is shown in the details panel. Due to this highly dynamic behavior, other interactions try to avoid moving the mouse across the map (as would be needed by a window menu, scroll bar or zoom slider) and instead are "local"– context menu, keyboard actions, and mouse clicks and drags. If desired, the current selection may be "frozen", in which case all nodes except the selected one are dimmed.

*Zoom, pan, traversal:* Dragging the map horizontally pans it, whereas a vertical drag zooms in and out. The system detects whether the drag is horizontal or vertical, modifies the cursor

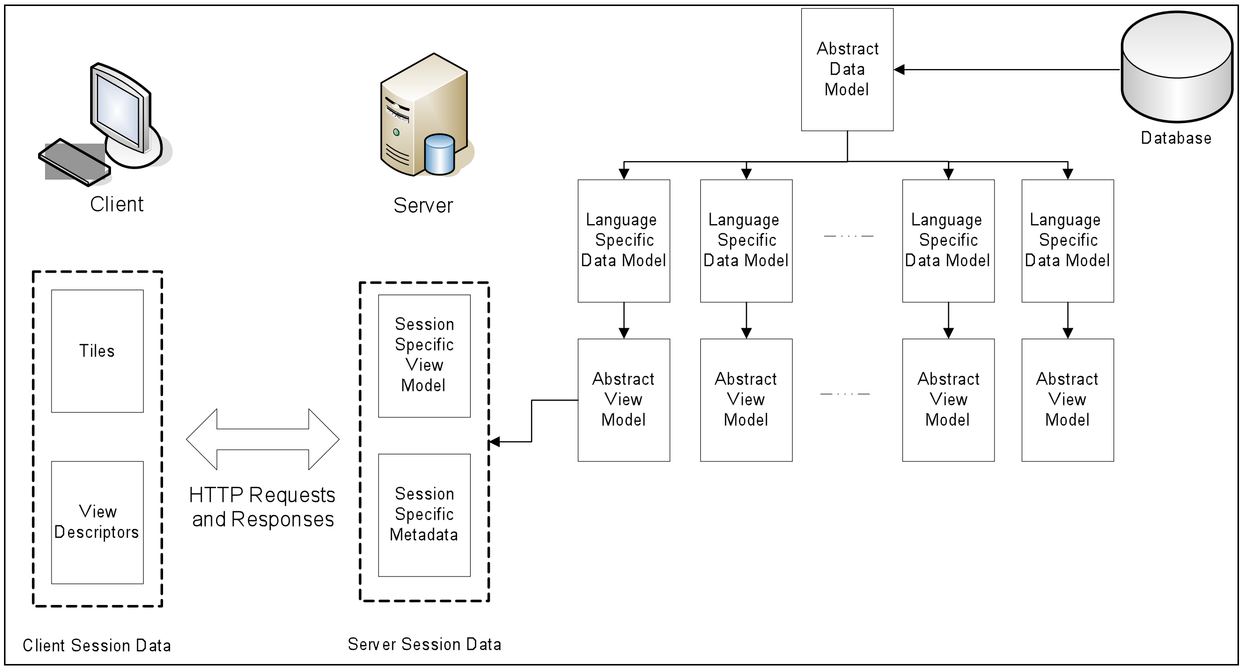

Client
Server
Abstract Data Model
Database
Language Specific Data Model
Language Specific Data Model
Language Specific Data Model
Language Specific Data Model
Tiles
View Descriptors
HTTP Requests and Responses
Session Specific View Model
Session Specific Metadata
Abstract View Model
Abstract View Model
Abstract View Model
Abstract View Model
Client Session Data
Server Session Data

**Figure 8: Client-Server architecture illustrating Client Artifacts and Server Models.**

accordingly, and proceeds with the operation. "Optimal zoom" via double-click or key press scales the selected node to occupy the entire available width (as shown for Jean Johnson in Figure 3), or wide enough for it and all its descendents to show faces.

*Search:* People can be searched for either globally or contextually (i.e., within the subtree rooted at the selected node) by various fields such as name or e-mail. Data from mashup sources can also be searched (e.g., finding patents whose title contains "speech" for a given department).

## 3. SCALABLE WEB SOLUTION

The Java system described in the previous section performs well for organizations containing up to around 30,000 people, once the organizational data is loaded. Startup time is reasonable for such sizes when reading the data from a local file system, but assembling and delivering the data from a server takes several seconds even for 1000 people. The architecture must therefore be rethought for scalability. We proceed to describe our scalable Web solution, which is dramatically faster for large organizations.

### 3.1 System Architecture and Overview

Our scalable Web architecture is illustrated in Figure 8. The data are maintained in a database organized for maximal efficiency. Since the data here exhibit a hierarchical organization, the database chosen is an LDAP server, and the data are stored in directories corresponding to their place in the hierarchy. For efficiency, 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 an *Abstract Data Model* to a *Session-specific View Model*, that is ranging from the most general to the most specific, respectively. They have been chosen to minimize memory footprint while maximizing shared data among users. The single largest model is the Abstract Data Model. It contains as much of the data from the LDAP server as practicable, and is shared among all clients. Further, it is natural-language independent, with language constructs represented by tokens that describe the data and related fields. 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.

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 with an arbitrarily large screen size of the client. This model contains only information on the positions of individual nodes within the view and lacks dependence on any 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

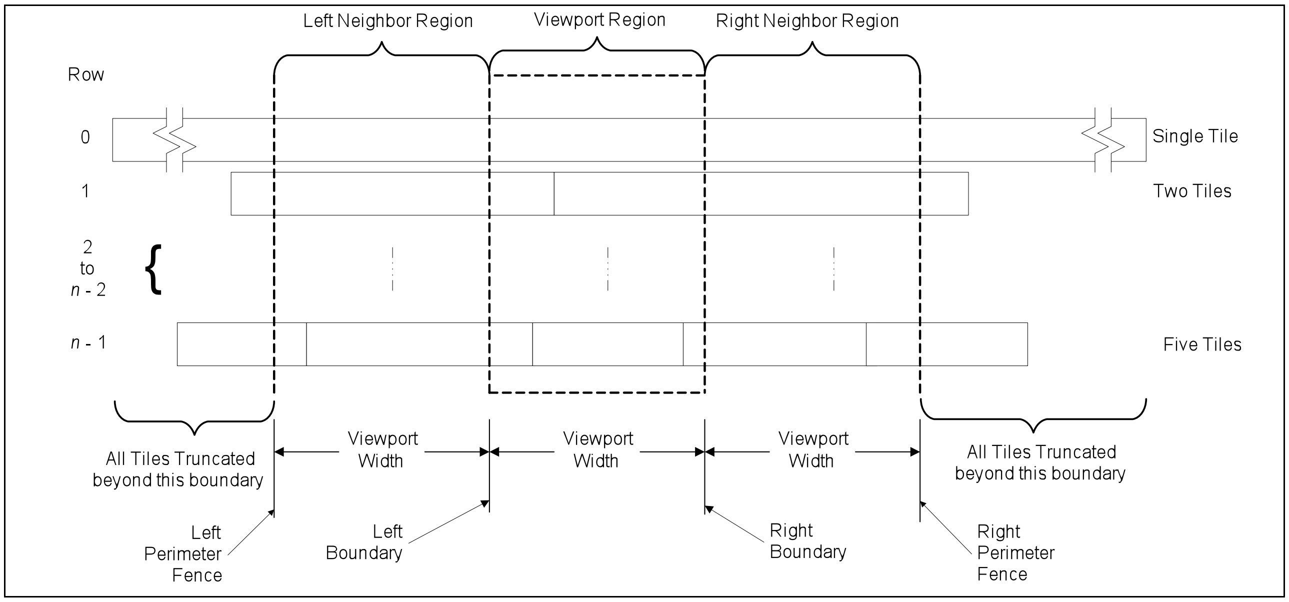


**Figure 9: Example of Tiles for Hierarchy of *n* Levels**

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.

The server performs a series of transformations upon the Session Specific View Model while preparing the visual images for the client. The first transformation corrects for the limited resolution of the client device. The session specific view model may represent a large dataset, so large that if it were to be viewed in total on the client, the leaf nodes would be smaller than one pixel. In order to rectify this, the server combines, or 'elides', such leaf nodes into larger blocks until they are large enough to be visible.

Additional metadata may be associated with an individual in the data set's model, such as a picture or label. Of course, it is only sensible to attempt to display such metadata 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 user may not wish to view the entire session specific view model as a single image. Indeed, for a large dataset, the resulting image may not convey the desired information, much like the case of a topographical map: when viewed in its entirety, a book-sized map of a large country will not yield much information about individual street names in the capital city. We view the width of the leaf nodes as an adjustable parameter, affording the user the ability to control the level of detail granted to a particular region of the view model. The width of the leaf nodes, or *zoom-level*, may assume a continuum of values, not just the discrete set familiar from topographical maps.

The server must take this zoom-level into account in several places. First, when the zoom-level is increased, it may no longer be necessary to elide leaf nodes as they will now be visible as separate entities. More significantly, 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. This tiling scheme is illustrated in Figure 9. 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. 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. Additionally, no individual in the organization belongs to more than one tile, as splitting an individual across tiles could lead to a highly disruptive flicker when the tiles are sequentially loaded.

Construction of tiles proceeds in the following manner. First, the server computes an offset to the left-hand position of the session-specific view model so that the region of the user's interest is centered within the viewport; this offset is maintained in the session-specific metadata. Next, the server examines each level or *row* of the hierarchy in turn. Beginning with the left-most individual in each row and proceeding to the right in sequence from sibling to sibling, the server locates the first individual located at least partially in the Left Neighbor Region (See Figure 10). The server then creates a new tile, adding this individual as its first member. It adds siblings in turn until it encounters a sibling wholly or partially within the Viewport Region. In a similar manner, it creates new tiles for other individuals within the Viewport and Right Neighbor Regions. The tile is given a set of coordinates detailing its width and position of its left edge in the client coordinate space. Finally, the tile is given its unique tile key

In addition to the images prepared for the client, the server prepares a limited set of descriptors, delimiting various regions of the images. The client has no *a priori* knowledge of the location of any such regions: it only displays images. First, the server prepares descriptors for the several rows of the hierarchy given the positions of the tops and bottoms of each row. Next, the server creates descriptors of the individuals immediately above, below and to the left and right of the region of interest. Note that a single visible image may represent many hundreds of individuals. It is inefficient for the server to deliver descriptors for all, and also unwarranted since the user may only interact with one individual at a time. As the user interacts with the image, by moving the cursor across the map, he or she may change the region of interest. Consequently, the server computes and delivers additional descriptors for the new region of interest.

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

When an individual is selected for the first time, the server prepares and delivers a set of descriptors of the location of the individual within the image, and that of his or her parent, children, and right and left siblings. Additional further detailed information on the selected individual, such as telephone and office numbers are also prepared. The bounding box of the selected individual is highlighted with a border of a contrasting color. The metadata belonging to this individual and any further detailed information is displayed in a separate pane.

When the user first requests a tree rooted at a particular individual, the resulting image is devised exactly to fill the client's viewport. If the requested tree is large, many of the individuals as represented in the image may be so small that they are practically invisible. This condition may be rapidly altered through a zoom operation, consisting in changing the level of detail presented to the user. Unlike ZUIs that maintain aspect ratio, a zoom within the icicle plot is performed by changing the width of a leaf node, as all nodes are of the same height. The level of detail relative to the selected individual may be increased --- by zooming *in* --- or decreased --- by zooming *out*. In either case, a new set of tiles is requested from the server, and new set of view descriptors are prepared and delivered. The server maintains the artifacts associated with the previous zoom level for a predetermined time should the user wish to next return to this previous level, i.e., 'undo' the zoom operation. It should be noted that a zoom-operation occurs only the horizontal dimension, with the vertical left unchanged.

Panning consists of dragging the image being viewed to the right or to the left. For the first image delivered, crafted exactly to fill the viewport, such an operation would not be sensible, and is prevented deliberately. However, after a zoom operation, the full view may be larger, indeed much larger, than the viewport, should the view be displayed in its entirety Through a *pan*, regions beyond the viewport are dragged into view. This operation is accomplished very efficiently by dragging the image as a single whole, including both the visible parts of the image and the invisible parts beyond the borders of the viewport. Note that the image may be tiled, but all tiles are dragged simultaneously. Further, we choose to allow the user to drag the image at most a distance equal to the width of the viewport in a single action. Thus, providing a smooth pan operation from the user's perspective requires only that prior to the start of the pan operation those tiles be present just to the right and left of the viewport. At the conclusion of the pan operation, when say the pointing device is released, new tiles are fetched, in anticipation of a subsequent pan operation.

The pan operation has a requirement of some subtlety when the user is viewing hierarchical data. If there are any labels, images or other metadata associated with a particular individual, particularly one near the top the hierarchy and therefore represented by a wide node, this information may well have moved beyond the viewport. In such cases, the client might request a new tile of the server, as detailed below, with the metadata displayed at the appropriate new position. To the user, the metadata will appear to 'snap' back into view.

This process of adjusting the position of the metadata risks triggering a large number of interactions with the server, causing a reduction of responsiveness as the server redraws the tiles. The adjustment process begins a specific time after the pan has ended; if the user initiates another pan before this time elapses, the adjustment is postponed until after the end of this new pan, and so on. In order to further minimize interactions with the server, once the adjustment process starts we classify each tile into one of six categories as indicated in Figure 10: *Übertile*, a tile extending at least partially into the left and right neighbor regions; *Left Neighbor Hidden*, a tile contained only partially within the left neighbor region; *Left Neighbor Visible*, a tile contained partially within the left neighbor region and partially within the viewport region; *Central Tile*, a tile entirely within the viewport region; *Right Neighbor Visible*, a tile contained partially within the Viewport and partially within the Right Neighbor Region; *Right Neighbor Hidden*, a tile only partially contained within the Right Neighbor Region; and *Hidden*, a tile entirely to the right or left of the corresponding neighbor region. The metadata of all Übertiles may be adjusted by the client alone without help from the server. We determine the tiles requiring adjustment in the following manner: tiles of types Hidden, Left Neighbor Hidden and Right Neighbor Hidden are completely invisible and need not be adjusted; a tile of type Central that was of another type before the pan operation will require adjustment, as will all tiles of types Left Neighbor Visible and Right Neighbor Visible. Existing tiles may also need to be adjusted if truncated. To ensure that they not extend beyond the left or right boundaries, their widths and positions are corrected so that they remain centered within the viewport. This correction is accomplished solely by the client.

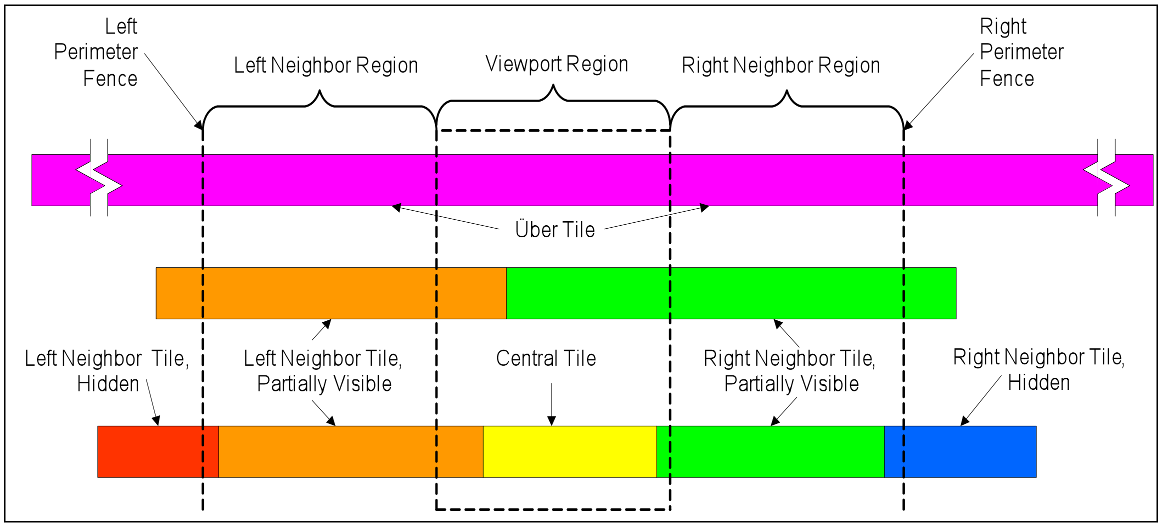

Left Perimeter Fence
Left Neighbor Region
Viewport Region
Right Neighbor Region
Right Perimeter Fence
Über Tile
Left Neighbor Tile, Hidden
Left Neighbor Tile, Partially Visible
Central Tile
Right Neighbor Tile, Partially Visible
Right Neighbor Tile, Hidden

**Figure 10: Classification of Tiles**

## 3.3 Implementation Setup Details

The server machine we employ operates under Microsoft Windows 2008 Server, a 64 bit operating system affording full access to all of the machine's 24 gigabytes of RAM. We employ IBM WebSphere Application Server 6.1, IBM Tivoli Directory Server 5.2 and Mozilla Firefox 3.52. Similar results would be obtained for other server products, and we have ensured that our system performs properly when used with the Microsoft Internet Explorer 7 browser as well as WebKit based browsers.

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

We collected feedback on OrgMaps through demonstrations of the Java prototype inside IBM and also at the Lotusphere conference. 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.

The Java version is feature rich, with many key-based interactions. Although impressive for demonstrations, we observed that new users found the interface confusing. We took this under consideration when building the Web-based version, by relying more heavily on mouse interactions and providing an easily-accessible "quick help" mechanism.

The levels in the hierarchy need not be of the same type; they only need to support a child-parent relationship. An early prototype of OrgMaps was used during the ACM Programming Competition 2008 with the levels representing continents, countries, universities, teams and individual competitors. Participants quickly gained top-down insights into the geographic distribution of the contest.

We have made several improvements from user suggestions. One suggestion was to keep the faces visible for all qualifying visible nodes, even when they are significantly off-center; this had a significant impact on the dynamic tiling architecture. Another comment was that faces that occupy the entire height of a node seem to visually break it into 3 nodes (left, face, right). In response, faces are now fully embedded in nodes (with margins on top and bottom). Another suggestion was to use heated object spectrum for scalar mashups, as an improvement for color-blind users. Upon experimenting we decided not to follow this suggestion, as the different hues, when combined with the face images create too much visual complexity. That said, this raised our awareness of the importance of adding user controls for tailoring the view. Another user wished to use OrgMaps to glean groups in calendar invitations. As a consequence, we recently added the capability to visualize a set of people based on their names or e-mail addresses to the Web-based version.

We have recently begun deployment of the new Web-based version that is the focus of this paper within IBM. We look forward to gaining future insights.

## 4.2 Performance

One of the most important considerations for the user of a Web application, or any application for that matter, is the amount of time required before the application is loaded and ready for operation. Another factor is the speed of response to the user. In our initial prototype implementations of OrgMaps we used an architecture whereby a complete description of the

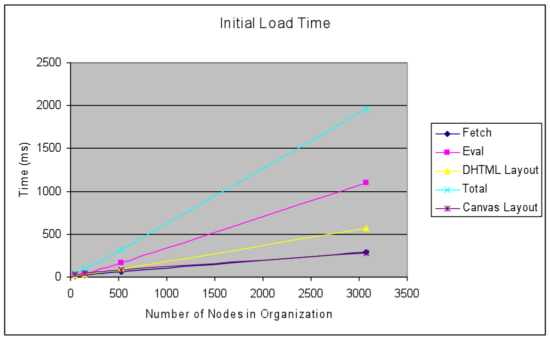


**Figure 11: Load Times for a client centric**

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

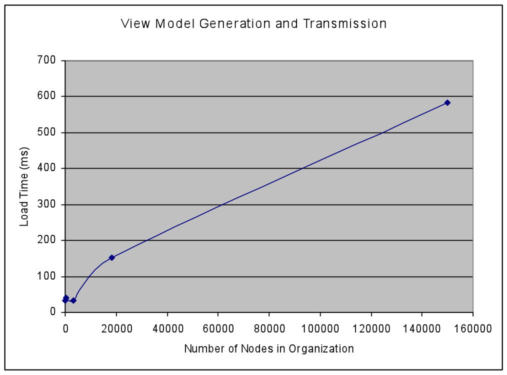


**Figure 12: Creation and Load Times for a scalable architecture**

11. It 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 11.

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 very much smaller set delivered to the client, as shown in Figure 8. This approach was detailed above 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 12. Note here that has proven possible to fetch organizations of as many as 149000 nodes with an acceptable response time of less than one second. Note that this time includes the layout time of Figures 11 since the layout and drawing of the image is performed by the server before it returns the set of descriptors to the client.

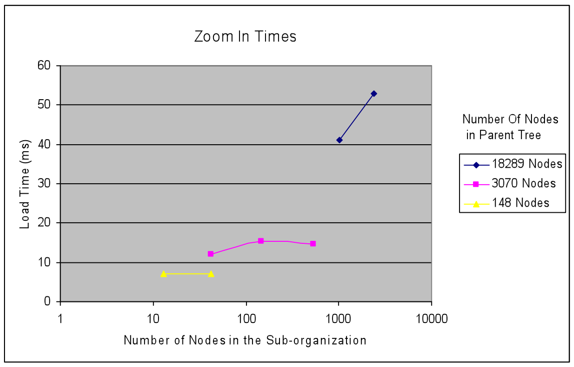


**Figure 13: Time to Create and Load *Tiled* View**

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 13 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 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. Through careful construction it proved possible to limit the Abstract Data Model for an organization of twenty-four thousand individuals to 34 megabytes; the size of the model is 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. If the user then zooms in on an individual near the top of the organization, creation of the resulting 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 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.

We had previously implemented several data mashup capabilities on the Java applet version. We plan to bring these existing capabilities to the Web-based version. Since the mashup is now done within the confines of a browser and not a native application, issues surrounding privacy and security of data arise. For example, if sensitive employee information that is available to a manager is mashed up we need to build confidence that the server does not have access to the data or if it does, it promptly destroys the user's data when the session terminates. Anonymization or hybrid techniques that send only intermediate data forms to the server may be necessary to protect the user's data.

We intend to improve the user interface to allow for more efficient use of the vertical dimension. If an organization is composed of many levels, each individual level may be very narrow. We have already implemented the ability to reroot the view at a lower level in the tree; an individual may be selected and a new view requested with this individual as the root. We also plan to extend the concept of a zoom to the vertical as well, allowing the user to select a subset of the available rows for display.

Some features we plan to consider center around social computing. Users may want to know how often and who in the enterprise is looking them up. Users may also like to be notified when the organizational structure or job responsibility of someone they work with changes. By maintaining logs of most popular people in the enterprise we can ensure that abstract view models for such individuals are prebuilt. Issues surrounding privacy clearly arise with such use cases and will have to be addressed on a case-by-case basis. In addition, we intend to experiment with pre-warming the models when we can predict that some may be particularly popular. For example, consider that some individual has just published an important paper; it should be possible to predict that many users would want to view this individual and his or her neighborhood.

Though memory utilization on the server is not a bottleneck at present, we plan to implement several memory management schemes to guarantee that the server memory is not exhausted after many user's activities. In particular, we will allow Abstract View Models, and Session Specific metadata to expire so as to free up memory. Similar techniques will be implemented on the client whereby view descriptors will be allowed to expire. As more Web applications are deployed, managing the browser cache will become an important aspect of controlling the memory usage of the browser. Interfaces allowing selective clearing of the cache will provide such control.

Our implementation is now also ready for wider deployments in the enterprise. We plan to make it available to every employee at IBM. Clearly, this deployment may lead to requests for additional features and surface issues that have not arisen hitherto.

## 6. RELATED WORK

Our work is principally differentiated from previous work in that, 1) it uses a basic Web client with no proprietary technologies, 2) all transmitted tiles reflect a specific user session, 3) its fluid, low-latency response necessitates a tiling of the visualization with look-ahead, 4) 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 5) we dynamically classify the view-specific tiles to reduce unnecessary communications.

The use of a range of client-server techniques for rendering display content in thin clients with different tradeoffs is old. In one extreme, in the VNC and GoToMyPC protocols the server sends only images to the thin clients. In the X Windows system, application logic runs on the server and rendering is done on the thin client (running the X Server), by sending it graphics primitives. With Citrix ICA and Microsoft RDP, rendering is done on the server and display updates are sent via a rich set of low-level graphics primitives to the client. SLIM [12] and THINC [1] also perform rendering on the server but take an intermediate approach to reduce bandwidth and latency by sending only a small set of higher-level graphics primitives that are transparently mapped to a few simple low-level primitives rendered directly by graphics hardware on the client. pTHINC [6] extend THINC to PDAs and performs server side scaling of images. However, none of the above techniques target specific Web-based applications, such as OrgMaps, with the need for dynamic and independent rendering tradeoffs within a browser.

Researchers in 3D computer graphics have encountered the challenges of handling large numbers of objects at interactive rates. Multi-resolution, view-dependent, and progressive mesh [5] representations of 3D models have been used to reduce the number of polygons that are sent to the rendering engine. Techniques, known as image-based rendering [16], have been devised to incrementally update images for small changes in viewpoints without having to render the complete model again. Chang & Ger [2] apply image-based rendering techniques to mobile devices and send rasterized images from a static scene with additional single-layered depth images to support occlusion-correct 3D rotation interaction. While much of this work was performed with native applications when the graphical models are available locally and not in the context of Web-based applications, the ideas are applicable to our problem. We have essentially used a hybrid approach that combines object-space and image-space rendering techniques. We compute view-dependent tiles in object space on the server and send down rendered image tiles to the client along with metadata to allow local view updates for pan operations without going back to the server. We also deliver the metadata progressively. With our design point we did not require data compression, which could increase the load on both the client and the server.

Another class of related work is those systems that use servers to perform rendering of complex 3D scenes for simple clients. For example, Poliakov et al. took this approach because of the lack of unified 3D APIs for the Web [11]. These frameworks have typically not supported very dynamic and interactive behavior. As is done with image-based rendering, the client morphs the cached data without a server round-trip to approximate a basic translation or rotation of the camera. See [9] for a more exhaustive set of related work and for an analysis of methods for remote visualization.

Most similar architectures to ours, such as those that display geographic data, allow zoom and pan only on static tiles. While Google Maps uses pre-rendered images that are session unspecific, Sorokine & Merzliakova [13] performs session-aware server-side rendering, but instead output simplified geometric primitives to be rendered client-side rendered in the browser. They also use limited navigation controls that do not perform look-ahead rendering for smooth navigation. D'Ambros et al. uses server-backings to tie multiple software engineering systems together providing a compact SVG output to the server [3]. They note the SVG data and rendering are performance inhibiting. Eick et al. [4] also use SVG output to Web clients from multiple server backings in a generalized visualization framework. Because of the use of SVG rather than rasterized images for the client, their performance suffers similarly to our initial client-rendered system.

Flashproxy [10] uses server proxies to interact with Flash content on devices that lack the run-time. They use SWF binary re-writing and a custom JavaScript bridge to permit interactions with the remotely-hosted SWF using only a basic Web client.

More widespread availability of WebGL will give browsers access to rendering hardware and could alter the tradeoffs that have been employed in web-based graphics and visualization.

Web-based mashups or service compositions was one of our motivations to make OrgMaps a Web-based service. Making mashups easy is a challenge. In [14], the authors provide a programming model for building mashups by end users and discuss the implementation and evaluation issues associated with end-user mashup programming. In [8] a framework that allows users to easily build mashups from within a familiar spreadsheet environment by making complex data as first class spreadsheet cell values is presented. The Marmite [15] tool for end-user programming on the Web works by displaying a linked data flow spreadsheet view, letting people see the program as well as the data simultaneously.

## 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] Baratto, R., Kim, L., and Nieh, J., THINC: A Virtual Display Architecture for Thin-Client Computing, In Proc. SOSP, 2005, pp. 277-290.

[2] Chang, C. and Ger, S., Enhancing 3D Graphics on Mobile Devices by Image-Based Rendering. In Proc,of the Third IEEE Pacific Rim Conference on Multimedia: Advances in Multimedia information Processing pp. 1105-1111.

[3] D'Ambros, M., Lanza, M., Lungu, M., and Robbes, R. Promises and Perils of Porting Software Visualization Tools to the Web. In Proc. of WSE 2009 (11th IEEE International Symposium on Web Systems Evolution).

[4] Eick, S. G., Eick, M. A., Fugitt, J., Horst, B., Khailo, M., and Lankenau, R. A. 2007. Thin Client Visualization. In Proc. of the 2007 IEEE Symposium on Visual Analytics Science and Technology, 2007, pp. 51-58.

[5] Garland, M., "Multiresolution Modeling: Survey & Future Opportunities," Eurographics State of the Art (STAR) Report, In Proc. Eurographics, 1999.

[6] Kim, J., Baratto, R., and Nieh, J., pTHINC: A Thin-Client Architecture for Mobile Wireless Web, In Proc. WWW 2006, pp. 143-152.

[7] 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.

[8] Kongdenfha, W., Benatallah, B., Vayssière, J., Saint-Paul, R., and Casat, F., Rapid Development of Spreadsheet-based Web Mashups, In Proc. of WWW 2009, pp. 851-860.

[9] Luke, E. J. and Hansen, C. D., Semotus Visum: a flexible remote visualization framework. In Proc. of the Conference on Visualization 2002 , pp 61-68.

[10] Moshchuk, A., Gribble, S. D., and Levy, H. M., Flashproxy: transparently enabling rich Web content via remote execution. In Proc. of the 6th Intl Conference on Mobile Systems, Applications, and Services 2008, pp. 81-93.

[11] Poliakov, A. V and Albright, E. M., Corina, D. P, Ojemann, G. A., Martin, R. F, and Brinkley, J. F., Server-Based Approach to Web Visualization of Integrated 3-D Medical Image Data. In Proc. American Medical Informatics Association Fall Symposium, 2001, pages pp. 533-537.

[12] Schmidt, B. K., Lam, M.S., and Northcutt, D., The interactive performance of SLIM: a stateless, thin-client architecture, In Proc. of SOSP 1999, pp 32-47.

[13] Sorokine, A. and Merzliakova, I., Interactive map applet for illustrative purposes. In Proc. of the 6th ACM international Symposium on Advances in Geographic information Systems, 1998, pp. 46-51.

[14] Wang, G, Yang S., and Han. Y., Mashroom: End-User Mashup Programming Using Nested Tables, In Proc of WWW 2009, pp. 861-870.

[15] Wong, J. and Hong, J. I., Making Mashups with Marmite: Towards End-User Programming for the Web, In Proc of CHI 2007, pp. 1435 - 1444.

[16] Zhang, C. and Chen, T., A Survey on Image-Based Rendering - Representation, Sampling and Compression," Technical Report AMP 03-03, June 2003, Electrical and Computer Engineering, Carnegie Mellon University.