RadMonitor: Radiology Operations Data Mining in Real Time

Richard Chen,¹ Pattanasak Mongkolwat,² and David S. Channin²

This paper describes the web-based visualization interface of RadMonitor, a platform-independent web application designed to help manage the complexity of information flow within a health care enterprise. The system eavesdrops on Health Layer 7 traffic and parses statistical operational information into a database. The information is then presented to the user as a treemap—a graphical visualization scheme that simplifies the display of hierarchical information. While RadMonitor has been implemented for the purpose of analyzing radiology operations, its XML backend allows it to be reused for virtually any other hierarchical data set.

KEY WORDS: Data mining, operations analysis, treemap visualization

BACKGROUND

The construct of a treemap as a “space-constrained visualization of hierarchical data” was pioneered by Ben Schneiderman in the 1990s.¹ The term is used to describe a viewing object that represents the big picture of thousands of nodes in a tree. Each node is a rectangle that contains the attributes of size, color, and color gradient. Recursive partitioning of a node’s rectangle among its children allows for a compact visualization of information.²

A popular example of this construct is SmartMoney.com’s Map of the Market.^{3,4} Information in this treemap is divided into blocks representing industries and stocks within an industry. The size of a stock’s rectangle is directly proportional to its market capitalization, and the size of an industry’s rectangle is a summation of its associated stocks. Similarly, the color and color gradient of a stock’s rectangle is related to its percent earnings growth or loss. A collage of these individual stock rectangles forms a patchwork color and color gradient hue representing the industry. This is reflective of the overall health within the sector.

Implementations of treemap web applications, such as the one seen on SmartMoney.com, using Java applet technology are relatively common. In fact, in a recent survey of visualization tools conducted by graduate students at the University of Maryland in the Spring of 2005, 9 out of 10 web applications using hierarchical display models relied exclusively on Java.⁵ The tenth was newsmap, a treemap implemented using Macromedia Flash.⁶ It displays news items in real time as polled from Google News, and uses the parameters of number of news sources to represent size, news category to represent color, and time elapsed since publication to represent the color gradient.

These diverse implementations reflect an increasing acceptance in the utility of treemaps. This interest is further supported by evidence that show an increase in the efficacy of user comprehension of hierarchical data through the use of such visualizations.⁷ Thus, the goal of RadMonitor was to extend the treemap construct into the hospital enterprise to support the analysis of radiology operations information. Rather than delivering a web-based solution through an external plug-in such as Java or Macromedia Flash, RadMonitor relies only on native, standards-based HyperText Markup Language (HTML), Extensible Markup Language

¹From the Ohio State University College of Medicine, 2343 Quarry Valley Road, Columbus, OH, 43204, USA.

²From the Department of Radiology, Northwestern University Medical School, 448 East Ontario Street Ste 300, Chicago, IL, 60611, USA.

Correspondence to: Richard Chen, Ohio State University College of Medicine, 2343 Quarry Valley Road, Columbus, OH, 43204, USA; tel: +1-773-4564714; e-mail: rickychen@gmail.com

Copyright © 2007 by Society for Imaging Informatics in Medicine

Online publication 30 May 2007
doi: 10.1007/s10278-007-9033-8

Fig. 1. This is a representation of the data flow across RadMonitor.

(XML), and JavaScript. It is the only solution that we are aware of at this time that exploits the recent advances in these technologies, and the result is a product that delivers a web experience comparable to the interactivity and responsiveness of a native desktop application. This paper will focus on the RadMonitor web interface, and specifically how the application can facilitate user analysis and interpretation of hierarchical radiology information.

MATERIALS AND METHODS

The RadMonitor design involves a traditional three-tier architecture consisting of the database,

server, and client (Fig. 1). Radiology operations information flows from a Health Layer 7 (HL7) data feed to a MySQL relational database. Clients then interact with this data through a JavaScript-enabled web browser and a server-side processing script.

In the backend, an HL7 application on the server receives order message and result message data feeds from Quovadx, formerly known as Cloverleaf. It is used to parse the HL7 messages, and allows us to map, route, and extract data from the hospital information system. Statistical information relevant to radiology operations management is stored in the RadMonitor database. Database entries include fields that represent order status changes, and start

Fig. 2. This is an illustration comparing the traditional web application model to the AJAX web application model.

Fig. 3. This is an illustration comparing the traditional interaction model versus the AJAX interaction model.

and stop dictation, and transcription time events. For the purposes of this paper, it is assumed that the database has already been populated.

The information in the database is available to the client to query and retrieve. This is done through a server-side processing script written in Hypertext Preprocessor (PHP). All of the exchanges between the server and client are done using HTML, XML, and JavaScript. While these technologies have been around for some time, RadMonitor makes extensive use of a relatively new twist of the technology—the XmlHttpRequest object.

Asynchronous JavaScript and XML

XmlHttpRequest is a JavaScript object invented by Microsoft (Microsoft Corporation, Redmond, WA, USA) in 1999.⁸ Since its appearance in Internet Explorer 5.0, it has been incorporated in nearly every other major web browser on the market, including Mozilla Firefox, Apple Safari, and Opera. The XmlHttpRequest object allows a client web browser to use JavaScript to asynchronously request information from a server (Fig. 2).

The server receives the request and responds to the client using XML. A second component of the XmlHttpRequest receives this response. The XML is then processed and rendered to the web browser, using HTML. Asynchronous JavaScript and XML, known simply as AJAX, is a term that was coined in early 2005 to describe this set of interactions.⁹ AJAX exchanges occur in a background process of the current web page, and are transparent to the end-user. This allows applications to respond dynamically without the need for a page refresh and without a gap in user interaction.¹⁰

In the traditional web user interaction model, data flow is largely unidirectional and synchronous (Fig. 3). A user clicks on a link or submits a form, and then has to wait for the system to process the request. The result requires a full-page refresh or page redirection to be displayed. In the context of a treemap, a user would be able to navigate to a site that displays a static treemap. Any requests for additional information about a particular node rectangle, however, would either have had to been sent along with the original treemap, or would have to require a full-page refresh

Fig. 4. This is the default view of RadMonitor. It is centered upon the treemap in the middle. User navigation tools are provided in the upper-right corner, while additional information is displayed in a panel on the right.

to be displayed. Either of these options would have been unacceptable for the user experience in our RadMonitor implementation. The first would have required an extraordinary amount of information to be sent with every client request, contributing to reduced application responsiveness. The second would have required a stop-and-go interaction model, which would have detracted from the interactivity of the application.

The asynchronous nature of AJAX has revolutionized the web user interaction model (Fig. 3). Google GMail and Google Maps are two popular examples of its real-world application.^{11,12} Google

Maps, for example, load only the minimum number of images necessary to display a map on the screen. Panning, zooming, and further interactions with the map cause additional map images to be loaded dynamically and on-demand in the background.

This seamless experience deviates from the confines of the traditional stop-and-go interaction model. RadMonitor makes full use of these AJAX technologies. The ability to make asynchronous requests allows us to limit the amount of information that the server needs to send to the client, and allows us to fetch and update data only when necessary. We are able display context-specific treemap node

Table 1. Subset of the Features Compared in Deciding Between AJAX, Java, and Macromedia Flash

Design Criteria	AJAX	Java	Macromedia Flash
Ease of dynamic content generation	Easy	Easy	Difficult
Interactivity features	Medium	Low	High
Page loading times	Fast	Slow	Fast
Platform independent	Yes	Yes	Yes
Programming language support	Moderate	Strong	Moderate
Open standards	Yes	Yes	Limited
Web browser integration	Native	Plug-in	Plug-in
XML support	Yes	Yes	Yes

Fig. 5. This is the orders treemap. Slight changes in the server-generated XML semantics allow us to produce a uniquely different treemap visualization.

rectangle information without having to preload the data, or without having to refresh the page. Thus, we gain improvements in application responsiveness and interactivity through this efficient utilization of data communication.

IMPLEMENTATION AND DISCUSSION

RadMonitor was implemented to improve the delivery and utility of the treemap construct through a web platform. Our goal was to build a lightweight, extensible, and standards-based solution, complete with the interactivity and responsiveness expected of a desktop application. We compared our past experiences with AJAX, Java, and Macromedia Flash using a set of preselected design criteria (Table 1). In evaluating these technology options, we decided that the combination of HTML, XML, and JavaScript would be preferable to the use of either Java or Macromedia Flash. Application interactivity was a key component in our design, and the use of AJAX allowed us to dramatically improve the user experience by deviating from the traditional web interaction model.

Initially, only HTML is rendered in the client web browser. This acts as an empty layout template, upon which subsequent AJAX queries fill respective sections of the page (Fig. 4). The first section to be loaded is the treemap itself. Please note that the data in the following figures have been generated

for demonstration purposes. All of the development work and testing was done within our department.

Treemap

The treemap is the centerpiece of our application. The three treemaps that RadMonitor currently supports are orders, radiologist, and staff (Table 2). These selections are made using the radioboxes in the upper right corner of the page. The radiologist treemap is the most similar to the Map of the Market. It is a representation of the radiologists' average dictation time. Information is divided into a hierarchy of modalities and radiologists within a modality. The size of an individual radiologist's rectangle is related to the number of studies that the radiologist has dictated. Correspondingly, the size of a modality's rectangle is indicative of the total number of studies dictated within that modality. In Figure 5, there appear to be fewer Nuclear Medicine (NM) exams dictated compared to the other modalities. This is, in fact, true (NM 12 studies, computed radiography 17 studies, magnetic resonance 18 studies, and computed tomography 20 studies).

The color and color gradient of a radiologist's rectangle is a measure of the average time that the radiologist has spent dictating exams compared to the modality average. Green represents an average dictation time faster than the modality average, and red indicates times slower than the average. A

Table 2. Interpretations of the Different Parameters Utilized in the Three RadMonitor Treemap Representations

	Radiologist Treemap	Staff Treemap	Orders Treemap
Individual node rectangle			
Size	The size of an individual radiologist's node rectangle correlates to the number of exams that he or she has dictated. The larger the rectangle, the more exams that the radiologist has dictated.	The size of an individual staff member's node rectangle correlates to the number of exams that he or she has transcribed. The larger the rectangle, the more exams that the staff member has transcribed.	The size of an order node rectangle correlates with the amount of time that the order has been in its current status. The larger the rectangle, the longer that the order has been in its current status.
Color	The color of an individual radiologist's node rectangle correlates with his or her average dictation time compared to the modality average. By default, green represents times faster than average, and red represents times slower than average.	The color of an individual staff member's node rectangle correlates with his or her average transcription time compared to the modality average. By default, green represents times faster than average, and red represents times slower than average.	The color of an order node rectangle correlates with the status of the order. RadMonitor supports the following default conventions: entered, black; ordered, red; started, orange; completed, yellow; dictated, green; transcribed, blue; final, purple; and cancelled, gray.
Color gradient	The color gradient of an individual radiologist's node rectangle correlates with the degree in which his or her average dictation time differs from the modality average. A bright green color represents times slightly faster than average, and a dull green color represents times slightly slower than average.	The color of an individual staff member's node rectangle correlates with the degree in which his or her average transcription time differs from the modality average. A bright green color represents times much faster than average, and a dull green color represents times slightly faster than average.	The color gradient of an order node rectangle is unused at this time.
Modality node rectangle			
Size	The modality node rectangle size is a summation of the individual node rectangle sizes. The larger the node rectangle, the greater the number of exams that have been dictated within the modality.	The modality node rectangle size is a summation of the individual node rectangle sizes. The larger the node rectangle, the greater the number of exams that have been transcribed within the modality.	The modality node rectangle size is a summation of the individual node rectangle sizes. The larger the node rectangle, the longer the individual orders have been in their respective status within the modality.
Color	The modality node rectangle color is a patchwork hue of the individual node rectangle colors. Appearances more green in color represent individual dictation times that are generally faster than the modality average.	The modality node rectangle color is a patchwork hue of the individual node rectangle colors. Appearances more green in color represent individual transcription times that are generally faster than the modality average.	The modality node rectangle color is a patchwork hue of the individual node rectangle colors. Appearances tending toward a single color represent a majority of orders with that respective status.
Color gradient	The modality node rectangle color gradient is a patchwork hue of the individual node rectangle color gradients. Appearances more bright green in color represent individual dictation times that are generally much faster than the modality average.	The modality node rectangle color gradient is a patchwork hue of the individual node rectangle color gradients. Appearances more bright green in color represent individual transcription times that are generally much faster than the modality average.	The modality node rectangle color gradient is unused at this time.

similar hierarchical breakdown is applied to the staff treemap, which displays information based on the staffs' average transcription time.

The use of these parameters results in a treemap that conveys information about the real-time operational statistics of the radiology department. This information is intuitively represented, and allows us to readily answer questions such as: Which modality reads the most number of exams? The least? How are the dictation times different between modalities? How about within a modality?

Whereas the radiologist and staff treemaps are represented by number of exams (size) and average times (color and color gradient), order information can be similarly grouped into a hierarchy. In Figure 5, for example, the orders treemap uses the parameters of time elapsed since the last order status change to represent size, and order status to represent color. An order that has just "started" is orange, an order that has just been "completed" is yellow, and so forth. This new and novel treemap semantic allows us to answer a completely different set of questions, such as: Which orders have taken the longest time to progress? What order status is associated with a bottleneck in performance?

RadMonitor simplifies the construction of these diverse visualizations as it calculates and renders the treemap dimensions on the client rather than on the server. AJAX initially requests the radiobox-specified treemap from a script on the server. In Figure 6, the user is requesting a treemap of the type "radiologist." The additional parameters will be explained in a later section.

Upon receiving the request, the server-side processing script queries the database for the requested information. In the case of a radiologist treemap, this includes finding the number of exams within each modality that are marked "dictated", and using these exams to calculate and compare between a modality and an individual radiologist's average dictation time. These calculations determine

the size, color, and color gradient values. The result is then returned to the client as XML, where it is processed asynchronously by the receiving component of the XmlHttpRequest. JavaScript code on the client uses the provided values to perform the layout calculations necessary to generate the treemap.

Thus, simple XML mark-up changes allow RadMonitor to display a diverse set of treemaps. There is a strict separation of the server-generated XML from the client-generated treemap. The versatility of this design facilitates RadMonitor's reusability. Furthermore, the XML exchanges involve only the minimum amount of information necessary to create a treemap. This bandwidth reduction correlates directly with an increase in application responsiveness.

Context Menu

RadMonitor includes a context menu associated with each individual node rectangle (Fig. 7). The menu appears automatically when the mouse hovers over an item. It provides immediate feedback on the data representations of size, color, and color gradient. This information is readily available as these parameters were all necessary for the original treemap layout calculation, and were thus part of the initial server XML response. In the case of the radiologist treemap, for example, the context menu displays a radiologist's number of read exams (size) and also his or her average dictation time (color and color gradient).

Whereas this information is insightful, it does not provide a significant level of detail. In this case, clicking on a node rectangle of interest will populate the context menu with additional information. This dynamic interactivity is provided through AJAX. The click action creates an asynchronous connection with the server. Identification parameters of the selected node are sent with the request, and the server responds with the context-specific informa-

Fig. 6. This is the query that the client sends to the server for a radiologist treemap, and a portion of the server-generated XML reply.

Fig. 7. Left: a small context menu with size, color, and color gradient information is available upon hovering over a node rectangle. Right: additional information is available by clicking on the node rectangle.

tion in XML-formatted HTML (Fig. 8). This is then processed and rendered in the context menu. On-demand information retrieval reduces the need to send a large amount of information. This detailed context menu information could have been included in the original treemap XML response. The cost, however, would have been a compromise in application responsiveness. Thus, at any point of interaction in RadMonitor, AJAX ensures that we are delivering a lightweight experience by eliminating extraneous data exchanges.

Navigation

In addition to the data navigation schemes provided by the treemap and context menu, RadMonitor also provides search functionality. Users are able to query the database for specific information of interest, such as patient name and/or study accession number. The results of a

submitted search query are then highlighted on the treemap, with the target node rectangles selected and displayed along with detailed information in their context menu. As with RadMonitor's other features, a search is accomplished without a page refresh. Its implementation contains a data-driven autocomplete component (Fig. 9). This is similar to that popularized by Google Suggest.¹³ As users type in an order accession number, for example, a drop-down box of potential matching numbers as listed in the RadMonitor database appears beneath the search textbox. This drop-down box is selectable and updates in real time, eliminating the need to type out the entire accession number. Thus, choosing items from the autocomplete drop-down list saves the user time while reducing clerical errors.

Autocomplete relies on the use of triggers in the search textbox. As the user types a query, every

Fig. 8. This is the query that the client sends to the server for additional context menu information, and a portion of the server-generated XML reply.

Fig. 9. This is the autocomplete feature.

change in the textbox string value creates an AJAX query to the RadMonitor database for a list of likely matching values. This list is then returned to the client as XML, where it is parsed and updated in the drop-down box in real time. The client-server communication appears seamless to the end-user as he types out his search query.

Side Panels

The default panel on the right side of the RadMonitor web page is the information panel (Figs. 4 and 10). This panel provides numerical information to complement the treemap visualization, and is meant to be an effective way to

support multidimensional data interpretation.¹⁴ It provides users with a quick overview of the current status of the entire department. The “at a glance” pane is a JavaScript chart implementation, listing the counts of the various order statuses at 10-min intervals for the past hour.¹⁵ The color of the chart line correlates with the color setting of the order status as specified in the user preferences (see below). There are four other areas of information in the information panel, the department averages, section averages, radiologist averages, and tech averages. Clicking on any of these headings will dynamically load a table of statistical averages. This table is sortable by any of its headers, “started to completed”, “completed to

Fig. 10. This is the information panel.

dictated”, “dictated to transcribed”, “transcribed to final”, and “total.”

RadMonitor also provides a number of features to allow user customization. These options include an advanced panel to diversify the set of information that the treemap can provide, and a preferences panel to customize the treemap interface. As with many of the other features, these functionalities have been designed to leverage the advantages of AJAX. Cookies are used to store these user settings, and the persisted values are given preferential treatment when interacting with the application. Links are available to restore the default values.

The advanced data customization panel is included to tailor the treemap data representation (Fig. 11). By default, a treemap includes all studies within the past 24 h in its search criteria. Available customization parameters include settings to specify a start time (ie 7 AM to 7 PM and 7 PM to 7 AM), a start date (ie last 24 h, last week, etc.), and/or a specific study status (ie entered, ordered, started, etc.). In addition, users can limit the total number of studies that they receive from the server. This is useful to improve application performance on older computers. Using the advanced panel, users in the radiologist treemap can, for example, specify data constraints such as showing all exams from the last week dictated

between 7 AM to 7 PM. These customized data requests are submitted as parameters in the Uniform Resource Locator (URL) of the AJAX treemap request (Fig. 6). The XML response with these settings is received from the server, and dynamically rerendered as a new treemap on the client.

The preferences panel allows user customization of the treemap display (Fig. 12). Options are available to change the node rectangle colors. Users unable to discriminate between green and red, for example, may prefer to view the treemap using blue and yellow. Options are also available to change the color and width of the node rectangle borders. Changes in any of these settings are passed as parameters in the URL of the Cascading Style Sheet. The style sheet is a PHP script that can dynamically propagate these values. Reloading the style sheet on the client web browser applies the specified changes. As with RadMonitor’s other features, these display changes occur transparent to the user.

Future Directions

Our implementation of RadMonitor as a lightweight treemap client provides a platform in which future improvements in accessibility and visualization can be easily incorporated. In our current implementation, we use a recursive

Fig. 11. This is the advanced panel.

Fig. 12. This is the preferences panel.

binary tree algorithm (known as the slice-and-dice treemap algorithm) to calculate a node's width and height relative to its parent node.¹⁶ Given an array of elements, the sum of the size parameters of the entire array represents the size of the parent node. This node is sliced in half to create two child nodes. The sizes of these nodes are determined by the sum of the size parameters in the first half of the array compared to the sum of the size parameters in the second half of the array. Each recursion alternates between slicing the parent node in half either horizontally or vertically. The resulting layout is thus determined by the position of the elements in the initial array. We are looking to replace this algorithm with a pivot or strip algorithm, which can calculate a layout based on a weighted parameter.^{17,18} This allows us to cluster together nodes with similar attributes. Research trials of these solutions have shown improved human recognition and performance benefits.

With regard to the treemap visualization, we are looking at ways to dynamically add, remove, or change nodes in the tree hierarchy without reloading the treemap. The challenge is in keeping the size and position of the other nodes in the data tree relatively intact such that the addition, removal, or change of an existing node does not confuse the end-user. This is difficult in our current layout algorithm, as the recursive layout functions are interdependent on the other nodes of the tree. A change in the size of one node, for

example, will necessitate changes in the size of each and every other node in the tree to accommodate the new proportions.

CONCLUSION

Information is the lifeblood of any business and health care is no exception. No single information system can manage the entirety of a health care enterprise of any significant size. RadMonitor is a tool designed to help manage the complexity of information flow within the hospital network. As with the dashboard concept, its utility lies in its use of proprietary or open data standards to interact and interface with other hospital information systems.

HL7 and digital image communication in medicine (DICOM) standards are critical to this process. Specifically, the data feeds are instrumental for the communication of nonimage- and image-based information, respectively. Furthermore, both HL7 and DICOM data feeds can serve as a source of valuable information about the status of operations in a department. A surprising amount of analytical information is contained within these messages. In addition to being vital to clinical and research activities, this information can be monitored, in real time, to provide immediate decision support for administrative and management activities. This paper describes

a novel implementation of graphically representing this information as a treemap using the latest web-based technologies.

REFERENCES

1. Human-Computer Interaction Lab: Treemap. Available at <http://www.cs.umd.edu/hcil/treemap/>. Accessed April 2006
2. Shneiderman B: Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans Graph* 11(1):92-99, 1992
3. Wattenberg M: Visualizing the stock market. In *CHI '99 Extended Abstracts on Human Factors in Computing Systems*. Pittsburgh, PA, 1999, pp 188-189
4. SmartMoney: Map of the Market. Available at <http://www.smartmoney.com/marketmap/>. Accessed April 2006
5. University of Maryland: Visualization Tools. Available at http://www.cs.umd.edu/class/spring2005/cmsc838s/viz4all/viz4all_a.html. Accessed April 2006
6. Google News: Newsmap. Available at <http://www.marumushi.com/apps/newsmap/newsmap.cfm>. Accessed April 2006
7. Turo D, Johnson B: Improving the visualization of hierarchies with treemaps: design issues and experimentation. In: *Proceedings of the 3rd Conference on Visualization '92*, Boston, MA, 1992
8. Wikipedia: XMLHttpRequest. Available at <http://en.wikipedia.org/wiki/XMLHttpRequest>. Accessed April 2006
9. Adaptive Path: Ajax: a new approach to web applications. Available at <http://www.adaptivepath.com/publications/essays/archives/000385.php>. Accessed April 2006
10. Adler S: WebOS: say goodbye to desktop applications. *netWorker* 9(4):18-26, 2005
11. Google: Google GMail. Available at <http://mail.google.com>. Accessed April 2006
12. Google: Google Maps. Available at <http://maps.google.com>. Accessed April 2006
13. Google: Google Suggest. Available at <http://www.google.com/webhp?complete=1>. Accessed April 2006
14. Kules B, Shneiderman B, Plaisant C: Data exploration with paired hierarchical visualizations: initial designs of PairTrees. In: *Proceedings of the 2003 Annual National Conference on Digital Government Research dg.o '03*, Boston, MA, pp 1-6, 2003
15. Chart: WebFX Chart. Available at <http://webfx.eae.net/dhtml/chart/chart.html>. Accessed April 2006
16. Asahi T, Turo D, Shneiderman B: Visual decision-making: using treemaps for the analytic hierarchy process. In: *CHI '95 Conference Companion on Human Factors in Computing Systems*, Denver, CO, pp 405-406, 1995
17. Bederson BB, Shneiderman B, Wattenberg M: Ordered and quantum treemaps: making effective use of 2D space to display hierarchies. *ACM Trans Graph* 21(4):833-854, 2002
18. Schneiderman B, Wattenberg M: Ordered treemap layouts. In: *IEEE Symposium on Information Visualization (InfoVis 2001)*, San Diego, CA, pp 73-78, 2001