MUVA: A FLEXIBLE VISUALIZATION ARCHITECTURE FOR MULTIPLE CLIENT PLATFORMS

Lea Skorin-Kapov*, Hrvoje Komerički§, Maja Matijašević§, Igor Pandžić§, Miran Mošmondor*

*Research and Development Center
Ericsson Nikola Tesla
Zagreb, Croatia
{lea.skorin-kapov}@ericsson.com

§Department of Telecommunications
FER, University of Zagreb
Zagreb, Croatia
nvr@tel.fer.hr

Abstract

Information visualization techniques are invaluable tools in numerous applications such as those involving large databases and document collections. Ubiquitous usage of such techniques can provide universal access to complex information. The main goal of our work is to provide such universal visual access to information through the proposed Multiplatform Universal Visualization Architecture (MUVA). MUVA is a collection of software modules that allow visualization of the same data across a wide range of platforms, from workstations to mobile phones, while automatically adapting the visualization and delivery modes to the particular platform. In the center of the architecture are the visualization tools, which represent various concepts of visualizing the data, such as 3D tree displays or simple 2D tables. The visualization tools are separated from actual client platforms by platform drivers, which adapt the output and delivery mode to each particular platform. On the input side, MUVA separates the data retrieval from the abstract visualization tools, so that any data source can be easily connected by implementing a thin application interface. In this way, MUVA can be used to provide ubiquitous information visualization for various services spanning a wide range of application domains. Implementation of the proposed architecture is demonstrated in two multiplatform visualization case studies; one for visualization of Grid monitoring data, and the other for weather data based on geographical location.

1 Introduction

Information visualization is a growing area of interest and research. As opposed to “traditional” data representations, user-friendly visualizations are becoming multi-modal, i.e., they may combine various media such as text, hypertext, pictures, multi-dimensional graphics, audio, and video in 3D Web-based virtual environments. Enhanced and interactive visualizations are designed to improve the ability of users to comprehend, work with, and interact with rich data spaces [4]. Various application domains that have been the focus of data visualization include large information spaces, software engineering, databases, network monitoring, data mining, and multi-user collaboration. It may be assumed that users, once being used to such interfaces, would wish to access such virtual environments by using various devices, for example, a powerful graphical workstation at work, a PC at home, and a Personal Digital Assistant (PDA) or a mobile phone while traveling. Recent advances in mobile and wireless networking, as well as improved presentation and communication capabilities of new mobile devices, have created a challenge to incorporate suitable visualizations in applications running not only on graphical workstations and PCs, but also on hand-held and wearable computers as well as new-generation mobile phones.

There has been a lot of recent work directed towards presenting content on mobile and hand-held devices. A software architecture to support computing on hand-held devices has been proposed in [9]. An overview of current technologies and standardization efforts for enabling device-independent Web applications may be found in [3]. One of the main technologies for adapting standard Hypertext Markup Language (HTML) content is transcoding [1], which unfortunately has known limitations when it comes to more complex structures. A recent approach to transcoding Web pages for display on mobile devices attempts to take into account semantics as well [7].

Still, applications designed from the start to run on multiple platforms are rather scarce and fairly recent. An example is a context-aware system for managing hospital information capable of running on a PDA and a desktop computer, presented in [10], which does not include visualization. Early work in complex visualizations on a PDA using remote visualization has been reported in [8]. To date, however, most applications that include visualization are still being developed to run on a target (group of) devices, rather than being open towards new devices and alternative visualization techniques.

In this paper we propose a Multiplatform Universal Visualization Architecture (MUVA). MUVA is a collection of software modules enabling visualization of data on multiple client platforms, where by *client platform* we consider a networking-capable terminal hardware (device) with specific operating system and application software. The focus of MUVA is not on applying visualization techniques to a specific application area, or on a specific platform, but rather on proposing a high-level flexible visualization architecture that may easily be applied to various domains. The key features of the proposed architecture are its independence of data acquisition, as well as a thin adaptation "layer" for the device on which the visualization is presented.

The paper is organized as follows. First, we briefly describe properties of typical client devices with their communication and presentation capabilities. Next, we describe the proposed architecture and its implementation. Finally, we demonstrate the application of the proposed architecture by two case studies. The first is a visualization of Grid monitoring data on four platforms: a desktop PC, a handheld PC, a Java-enabled new-generation mobile phone, and a standard WAP-enabled mobile phone. The second is a visualization of weather data on three platforms: a desktop PC, a handheld PC, and a WAP-enabled mobile phone.

2 Client platform capabilities

With the growing heterogeneity arising from differences in client network devices and network connections, there is a need for systems providing users with transparent access to services independent of access capabilities. Service content therefore needs to be adapted accordingly. Various parameters dictate client platform capabilities (e.g. available memory, processor speed, graphics card, display) and connection types (e.g. LAN, Wireless LAN, General Packet Radio Service (GPRS)). Table 1 gives an overview of some client devices that were used in this work as test platforms, with their respective characteristics, processing capabilities, and communication capabilities.

For the purposes of this paper we divide client platforms with regards to visualization capabilities into three groups (full, midi, mini) as follows:

- **Full clients:** platforms with enough processing power and visualization capabilities to locally process the raw data received from a server and visualize it while simultaneously supporting all visualization modes (2D and 3D graphics, tables, text, etc). In addition, the full client may offer better interaction capabilities with the data. Although other types of full clients may emerge, the representative of a full client is a PC (Windows, Linux, etc.) running a standard Web browser with Java support. Example interfaces and visualization tools may be implemented as Java applets (e.g. Shout3D) and standard HTML elements, so there is no need to download/install any software on the client machine. Additional software, however, may be installed if needed depending on the interface implementation (e.g. Virtual Reality Modeling Language (VRML) [1] plug-in). Client hardware and software may vary from lower-end configurations to higher-end configurations.

- **Midi clients:** platforms with medium processing power and visualization capabilities, powerful enough to present a user interface, simple 2D/3D graphics, as well as text and tables. The representative of a midi client would be a PDA (e.g. Compaq iPAQ or Palm) or a PDA-integrated mobile phone (e.g. Sony Ericsson P800).
- **Mini clients:** platforms that have insufficient processing power for local processing of raw data, and insufficient presentation capabilities for showing graphics. Such a terminal would receive pre-formatted WAP pages ready for presentation instead of raw data.

Table 1. Overview of processing and communication capabilities for client platforms

		Full Client PC (low-end)	Full Client PC (high-end)	Midi Client PDA	Midi Client Smart Phone
General Characteristics	Terminal Name	Desktop PC low-end	Desktop PC high-end	Compaq iPAQ 3870	Sony Ericsson P800
	Operating System	Windows 2000/XP	Windows 2000/XP	Windows Pocket PC (CE 3.0)	Symbian 7.0
	Browser	IE 5.5 / Netscape 6.0	IE 6.0 / Netscape 7.1	IE 3.02	Opera 6.0 / SE R101
		HTML, VRML, Shout3D	HTML, VRML, Shout3D	HTML, VRML, Shout3D	HTML WML
Processing Capabilities	Processor Type	PIII 800 MHz	P4 2.66 GHz	Intel Strong ARM SA 1110 206 MHz	ARM 9 156 MHz
		1066 MIPS	4904 MIPS	235 MIPS	N/A
	Memory Size	128 MB	512 MB	64 MB	16+16 MB
	Display Size	1024x768	1280x1024	240x320	208x320
	Color Depth	32-bit	32-bit	12-bit	12-bit
	Sound Quality	16-bit	16-bit	8-bit	16-bit
		Stereo	Stereo	Mono speaker, Stereo headph.	Mono speaker, Stereo headph.
Communication Capabilities	Network Type	LAN 100 Mbps	LAN 100 Mbps	GPRS CS-2 53.6 kbps / WLAN 11 Mbps	GPRS CS-2 53.6 kbps
	Latency	<10 ms	<10 ms	1s (GPRS)	1s
	Jitter	<1 ms	<1 ms	N/A	N/A
	Packet Loss	<1%	<1%	N/A	N/A
	BER	$<10^{-8}$	$<10^{-8}$	10^{-5} / 10^{-5}	10^{-5}

In addition to service customization based on client processing and communication capabilities, customization may be introduced through user preferences. For example, a user may wish to filter some media types (e.g. sound, video, etc.) in order to increase response time from the system and thus increase the functionality of a specific application interface at the expense of presentation level.

3 MUVA system architecture

In this paper, we propose a high-level visualization architecture called MUVA, which is capable of providing universal visual access to data independent of the client platform, while automatically adapting delivery modes to the particular platform. Rather than developing data visualization applications designed to run on a target (group of) client platforms, we propose to separate the platform adaptation procedure on the output side from the implemented data visualization technique, and so facilitate ubiquitous client access. In addition, by separating the data source on the input side from the visualization technique, the result is reusability of such techniques across a wide range of application domains offering different data sources. For example, the implementation of a technique such as a 3D tree representation of a hierarchical structure may be reused when developing any application used to visualize hierarchical data. For 2D visualizations, common graphical representations such as line graphs, trees, bar-charts, pie-charts, etc. may be used.

MUVA has been designed as a flexible and modular architecture comprised of a collection of software modules. Crucial parts of the architecture are the visualization tools, which represent various modes and concepts of visualizing the data (e.g. hyperbolic tree display; cone tree display; simple table; graph; text). Visualization tools are separated from actual client devices by platform drivers, designed to adapt the data delivery mode to specific platforms. On the input side, actual data collection is separated from the abstract visualization tools. This allows for any data source to be connected simply by developing thin application interfaces. The result is quick adaptability to various specific application domains. A service logic layer provides the necessary intelligence for connecting application interfaces, visualization tools, and platform drivers depending on the identified client platform capabilities and user request.

Figure 1 presents a conceptual view of the MUVA architecture. Each component of the architecture contains several modules, where not all of them have to be used in each application. A more detailed description of MUVA components is given below.

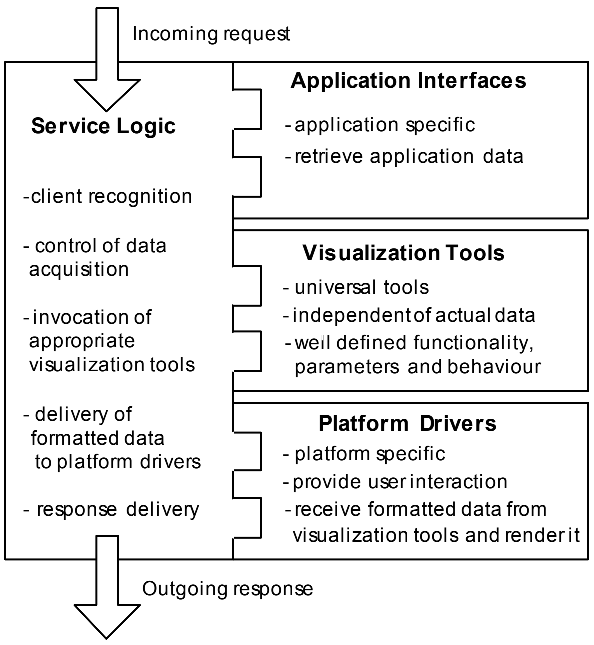

Incoming request
Service Logic
-client recognition
- control of data acquisition
-invocation of appropriate visualization tools
- delivery of formatted data to platform drivers
- response delivery
Application Interfaces
-application specific
-retrieve application data
Visualization Tools
- universal tools
-independent of actual data
-well defined functionality, parameters and behaviour
Platform Drivers
- platform specific
-provide user interaction
-receive formatted data from visualization tools and render it
Outgoing response

Figure 1. MUVA concept

Visualization tools are responsible for one particular mode of visualization (tool), e.g. a 3D structure of an input hierarchy, a simple table, pie-chart, etc. Each tool is standardized. This means that (1) it has a number of standard data parameters that can be fed to it through its API and (2) it has standard requests it can receive through its request interface, and what kind of visualization result it produces in reply to each request. For example, the parameters of a 3D Cone tree data display [12] may be the number of levels, number of children and their connectivity, specific alphanumeric data for each tree element etc. Valid requests to the Cone Tree tool may be: retrieve the Cone Tree as VRML; retrieve alphanumeric data for a specific node, etc.

Platform drivers are implemented for each supported platform. They render (visualize) formatted data received from visualization tools on the screen, and enable user interaction. This rendering may range from very simple (e.g. display a textual table) to complex, involving local interaction (e.g. display a complex VRML model with built-in interaction logic). Implementations may vary widely: for example, browser-embedded Java applets, standalone C++ or Java applications, even script languages. The communication with the visualization tools may be networked or local, depending on the location of the visualization tool in relation to the platform driver. Both may in certain cases be located on the server side, and in other cases on the client.

Application interfaces are responsible for retrieving data from a data source via a standard API. This API is application specific and must thus be tailored for each particular application. Retrieved data is converted to XML format based on the specified input interface for visualization tools. An example would be an application interface retrieving network monitoring data and converting it to XML to be used subsequently for populating a Cone Tree display with application-specific data.

Service logic encompasses modules that provide the intelligence needed to connect components of the architecture in order to enable universal visual access and delivery mode adaptation. Upon receiving a client service request, the client's preferences and platform capabilities are identified. One method of identification is based on the *User-agent* field in the HTTP header request. This approach, however, lacks parameters needed for forming a complete client profile (including access network characteristics, precise terminal characteristics, and user preferences). A more advanced method is based on Composite Capabilities / Preference Profile (CC/PP), a W3C Recommendation and a proposed industry standard for describing delivery context. The client profile data format is based on the Resource Description Framework (RDF). A client profile may either be sent directly as an extension to an HTTP request, or referenced from a remote location via URL. The implementation of client identification based on a set of generic profile parameters allows for on-the-fly identification of the capabilities of an increasing number of end-user devices.

The service logic retrieves raw data independently of the platform capabilities through invocation of application interface modules. The raw data is then sent to appropriate visualization tools. Formatted data received as the output from visualization tools is then delivered to necessary platform drivers. The service logic layer provides the logic necessary to accordingly select visualization tools and platform drivers to produce the final adapted content.

The modular design and separation of MUVA components allows for easy addition, modification, and maintenance of software modules. For example, the addition of a visualization tool would require only slight modification to the service logic (e.g. in terms of the conditions under which this tool would be invoked).

In order to demonstrate the proposed approach we show two services as case studies. In Section 4, we present a service providing a multiplatform visualization of Grid monitoring data, and in Section 5 we describe a service for multiplatform visualization of current weather data based on geographical position. Service implementation helps to demonstrate the separation between application interfaces, visualization tools, platform drivers, and service logic components, as well as the main communication channels involved.

4 Case study 1: Grid monitoring data

This service, as implemented in this work, provides an interface for users to view network configuration and monitoring data, such as load, data rates, and memory on different platforms. The data source used was a central data repository provided by the MonALISA system [11]. The MonALISA system provides a distributed monitoring service and was in this case used to monitor hundreds of AliEn Grid sites (http://alien.cern.ch/). Grid technologies have been described as supporting the sharing and coordinated use of diverse resources in distributed "virtual organizations" – that is, the creation, from geographically and organizationally distributed components, of virtual computing systems that are sufficiently integrated to deliver desired quality of service [6].

Our goal was to implement a Web based service that would provide a user with a view of the monitored network hierarchy, in addition to the values of various monitoring parameters for different Grid sites. Sites are organized into a hierarchy of "farms" and "clusters", referring to the geographical and/or logical grouping of nodes into virtual computing systems. The service presents an alternative to the existing MonALISA graphical user interface (http://monalisa.cacr.caltech.edu/) by providing support for multiplatform visualization and introducing new visualization techniques.

Users access the service by entering a unique URL, independent of the device being used. Requested data, which is then retrieved from a central repository and described using XML, is dynamically converted to a format suitable for displaying on the client device. The different formats that were used include VRML, HTML, and WML. The service implementation was tested on four platforms: a desktop PC, a handheld PC, a Java-enabled PDA-type mobile phone, and a standard WAP-enabled mobile phone.

Where possible, the monitored network configuration (or a particular sub-configuration) was visualized using the 3D Cone Tree technique [12]. Cone Tree is an interactive visualization technique suitable for hierarchical structures. The user interface is enhanced by enabling interactive viewing, zooming, expanding and collapsing of parts of the structure. A formal user study using a cone-tree-based file system visualization showed that although Cone Trees are not suitable for all tasks, users "were enthusiastic about the cone tree visualization and felt it provided a better 'feel' for the structure of the information space" [5]. The root of the network hierarchy is located at the tip of a semi-transparent cone. When a level in the hierarchy is expanded (on user click), its children nodes are distributed at equal distances around the base of a cone.

By applying this idea to visualization of Grid monitoring data, we visualize the hierarchy of farms, clusters and nodes as shown in Figure 2.

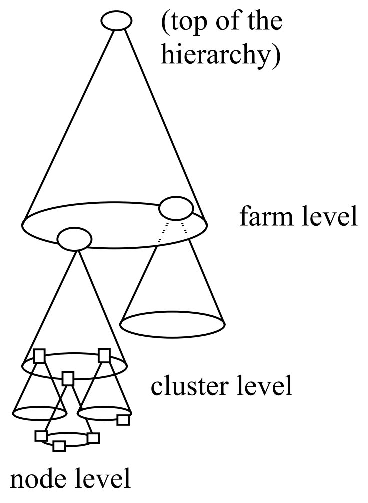


Figure 2. Grid hierarchy as related to a cone tree representation

The cone tree is interactive in that it also allows the user to vary the level of detail for a particular part of the hierarchy by expanding (or collapsing) a part of the cone tree. Monitoring data can be added to this graphical representation as well: for example, the load of a particular node (cluster, farm) may be shown by coloring the object representing that node (cluster, farm).

4.1 Implementation

In this section we describe the implementation of the MUVA system architecture modules shown in Figure 3, and discuss how each module fits into one of the following four components: service logic, application interfaces, visualization tools, and platform drivers.

4.1.1 Service logic

Client recognition functionality was implemented using Apache server 2.0.47. Due to fact that the developed prototype service was intended to be made available to users outside of a laboratory environment, and the current lack of widespread availability of CC/PP compliant terminal browsers, implemented client recognition is in this case based simply on identification of the browser type specified in the User-Agent header field of the HTTP request.

Data is retrieved in standard XML format through invocation of application interface components and formated depending on the platform capabilities and requests of the client. Requests are directed towards a Java servlet that is run using Apache Tomcat Server 4.1.27-LE-jdk14. The servlet requests data, and invokes the necessary visualization tool. In cases when 3D content is generated, the Cone Tree tool is called. Once the VRML result is received, the servlet calls on a platform driver to further adapt the VRML file for rendering in a Shout3D (Eyematic Interfaces Inc., www.shout3d.com) applet, after which the HTTP response is sent to the client.

In cases where the content generation is based on HTML or WML format, the servlet passes retrieved XML data to Apache Cocoon 2.0.4. Data is formatted into HTML tables or histograms using Extensible Stylesheet Language Transformations (XSLT) technology. Where necessary, additional platform drivers are invoked to further adapt the format prior to sending the response to the client.

4.1.2 Application interfaces

Data is always requested by application interfaces and returned in standard XML format. The interface towards the actual data repository storing monitoring data collected by the MonALISA system is based on Web Service technology. Connectivity to the Web Service was provided using Apache Axis 1.1 open source solution, the follow up on the Apache SOAP project. The stub code for the Web Service was generated by Axis' WSDL2Java utility and modified according to our needs. The Web Service returns values in the form of Java beans, that are then transformed by a *data format transformation* module to XML format.

4.1.3 Visualization tools

Once data is retrieved, visualization tools are needed to generate the actual data representation. Various visualization techniques were used, including text, 2D graphics, and 3D graphics.

The creation of a VRML Cone Tree display based on an input hierarchy was implemented using Java. In general, any form of hierarchical data structure may be given as a valid input. Additional requests to this tool may be to display a simpler Cone Tree (e.g. when the client device is a PDA), where certain elements of the tree hierarchy are filtered based on specific parameters (e.g. display only nodes or clusters belonging to a particular farm). In certain cases, due to terminal incapability to display or render complex 3D graphics, data is transformed to simple HTML tables or histograms by using XSLT.

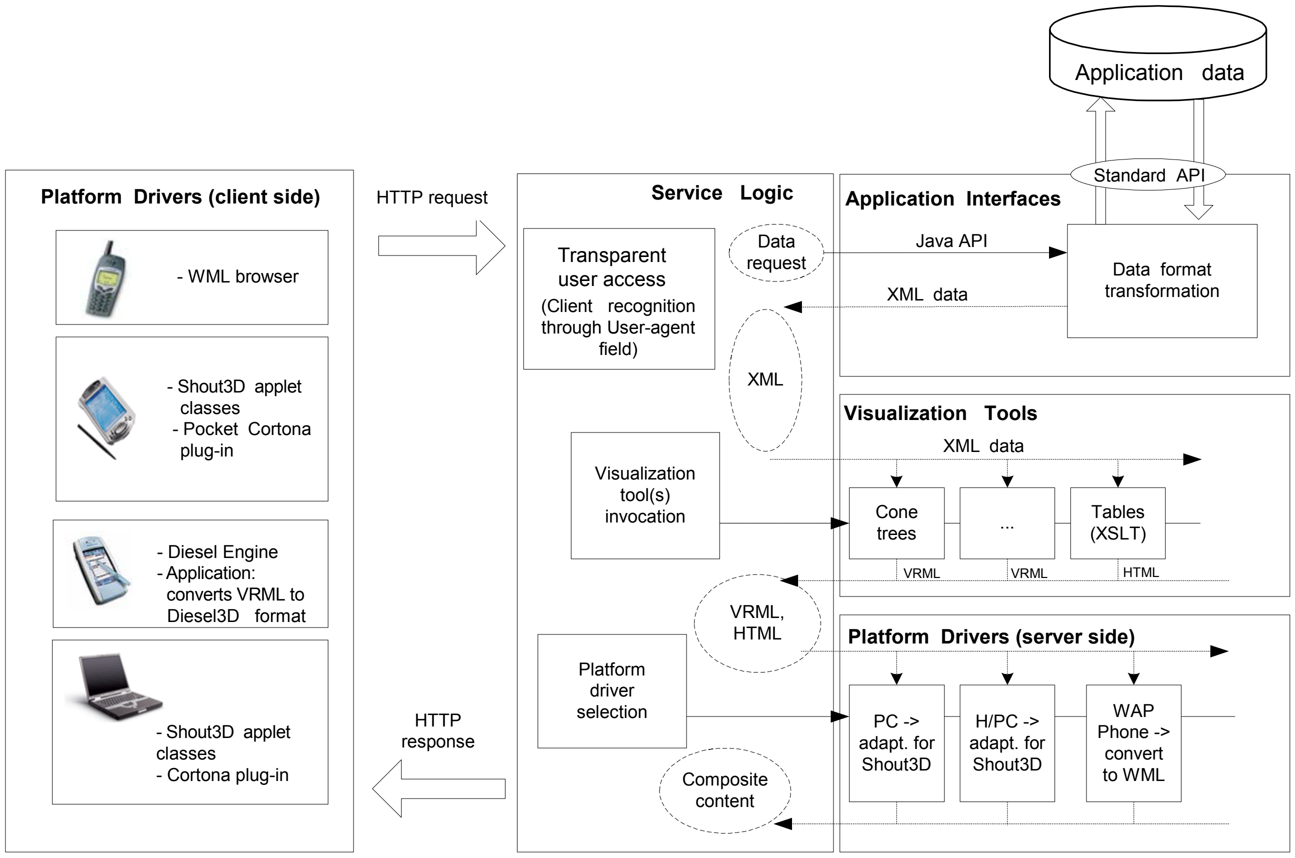


Figure 3. MUVA system architecture

4.1.4 Platform drivers

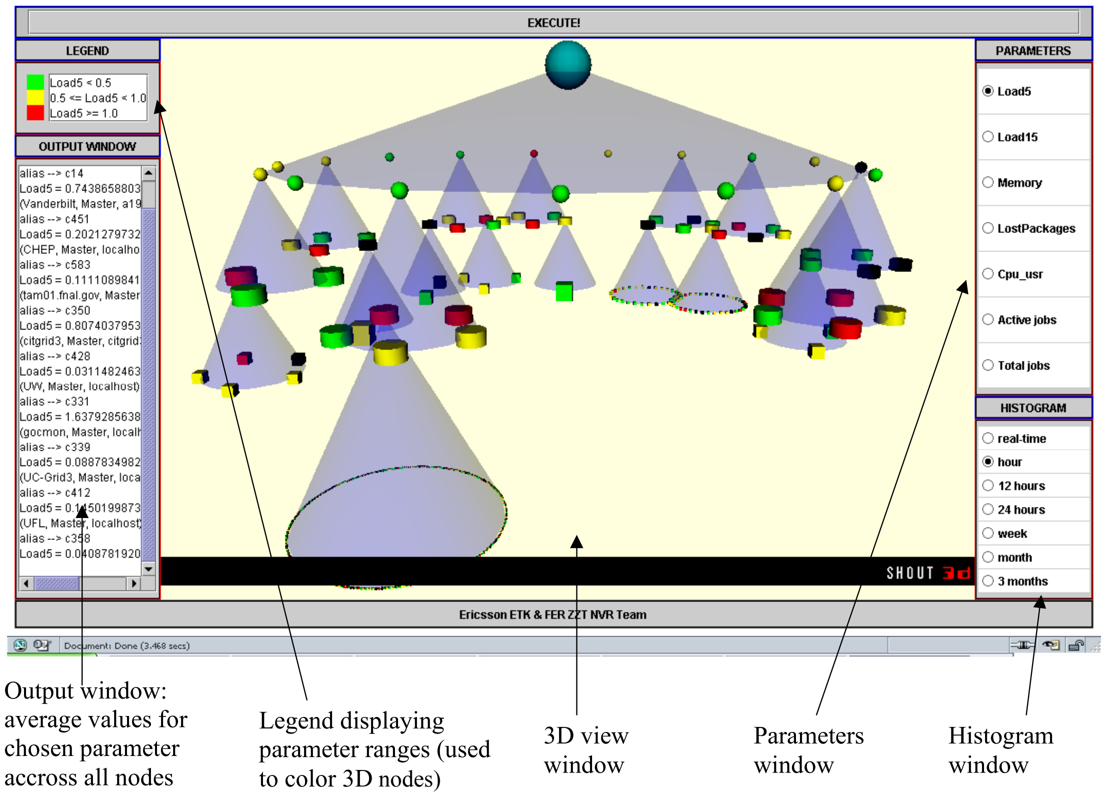
The interface displaying the 3D scene, designed to be viewed on a client with a standard Web browser and Java support, was implemented as a Java applet and based on the Shout3D engine. Shout3D is a library of Java classes for rendering 3D scenes over the Internet, thus offering the user the ability to view and interact with 3D scenes without the need for any additional plug-ins. In a different set-up, Cortona VRML plug-in and Pocket Cortona (for rendering on the iPAQ PDA) were used to render the 3D scene. Within the scope of MUVA, the Shout3D applet classes and Cortona plug-in are all considered platform drivers.

In addition to displaying 3D content on the PC and iPAQ clients, we implemented a C++ application to dynamically generate a 3D scene (in our case a 3D Cone tree display) on the Sony Ericsson P800 mobile device. The DieselEngine SDK 1.3 was used for software support. It is a collection of C++ libraries for creating 3D applications on mobile devices. Additional software used included Symbian UIQ v7.0 SDK and Metroworks CodeWarrior for Symbian OS. The application that was built reads a VRML file dynamically generated by the Cone tree visualization tool, parses the file, converts it to Diesel3D scene format and displays the content to the user. Additional interaction modules for navigation, camera manipulation, and object selection within the scene were also implemented. Also, XSLT files were implemented to further adapt content for display on a particular device. This includes creation of WML format for display on a WAP-enabled mobile phone.

4.2 Results

The result is a multiplatform Grid data visualization service that is developed in a modular fashion with the possibility for adaptation across different application domains. The implemented service provides visualization of monitoring parameters for a large number of Grid nodes. The monitored nodes are arranged in a hierarchical manner into farms and clusters. The MonALISA system collects monitoring data from all distributed sites and stores the data in a central repository. Data is collected by our service from the central repository via a Web Service interface.

A view of the *full client* display interface is shown in Figure 4. Upon initial loading, the *3D view window* renders the 3D scene displaying the dynamic node configuration. The imaginary top of the hierarchy is shown as a sphere, as well as the farms (actual top level in the Grid organization). Clusters are shown as cylinders, and individual nodes are shown as cubes.



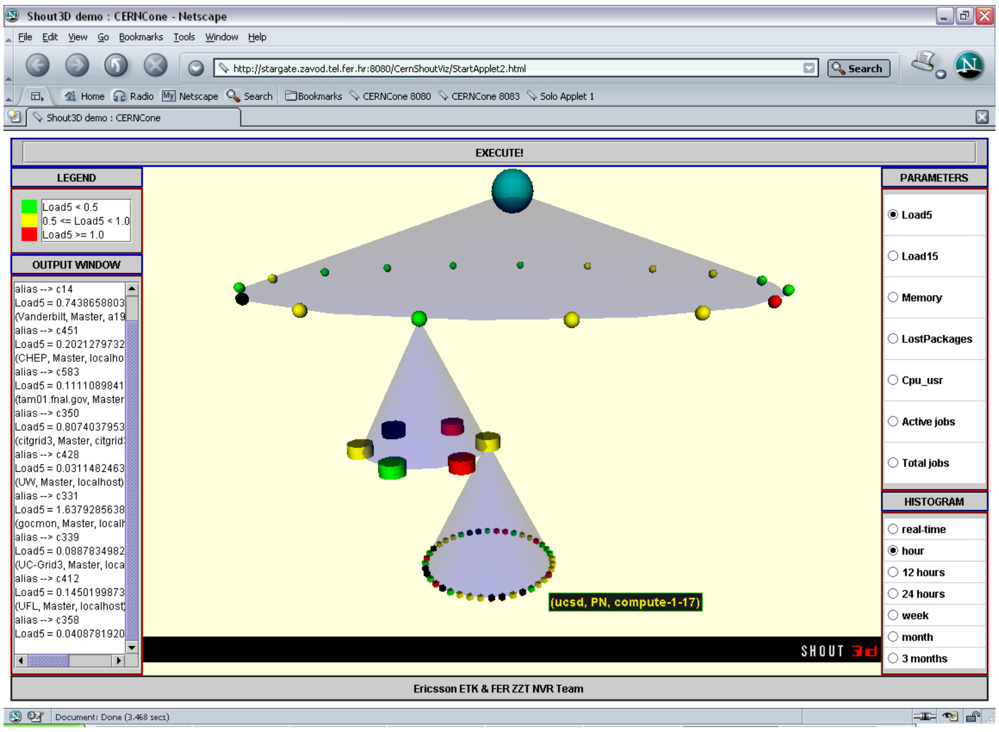
🟩	Load5 < 0.5
🟨	0.5 <= Load5 < 1.0
🟥	Load5 >= 1.0

```
alias --> c14
Load5 = 0.7438658803
(Vanderbilt, Master, a19
alias --> c451
Load5 = 0.2021279732
(CHEP, Master, localho
alias --> c583
Load5 = 0.1111089841
(tam01.fnal.gov, Master
alias --> c350
Load5 = 0.8074037953
(citgrid3, Master, citgrid
alias --> c428
Load5 = 0.0311482463
(UW, Master, localhost)
alias --> c331
Load5 = 1.6379285638
(gocmon, Master, localh
alias --> c339
Load5 = 0.0887834982
(UC-Grid3, Master, loca
alias --> c412
Load5 = 0.1450199873
(UFL, Master, localhost)
alias --> c358
Load5 = 0.0408781920
```

Selected: [✓ Load5], [✓ hour]

Figure 4. *Full client*: Java applet with text and 3D content

The *Parameters window* enables a user to choose a monitoring parameter. The *Histogram window* enables a user to choose between displaying real-time data and history data. Once the user has chosen a parameter and histogram button, clicking on the “Execute!” button will initiate the coloring of tree nodes and writing text to the output window. Parameter values are retrieved for each node, and coloring is based on the range that the value fits into. In Figure 4, it may be noted that overloaded clusters are colored red, medium loaded clusters are colored yellow, while underloaded clusters are colored green. The upper and/or lower threshold values for each range are displayed in the *Legend window*. The interactivity of the display is very useful when the user is interested in a particular part of the structure (for example, the status of nodes in the local computing center) and expands only that part of the hierarchy for further observation (Figure 5). Hiding some parts of the hierarchy, as well as selecting the parameters of interest, helps the user acquire an intuitive perception of the overall Grid infrastructure and status and at the same time allows a closer inspection of the part or parameter of interest.


Shout3D demo : CERNCone - Netscape
File Edit View Go Bookmarks Tools Window Help
http://stargate.zavod.tel.fer.hr:8080/CernShoutViz/StartApplet2.html
Search
Home Radio Netscape Search Bookmarks CERNCone 8080 CERNCone 8083 Solo Applet 1
Shout3D demo : CERNCone
EXECUTE!
LEGEND
OUTPUT WINDOW
PARAMETERS
HISTOGRAM
(ucsd, PN, compute-1-17)
SHOUT 3d
Ericsson ETK & FER ZZT NVR Team
Document: Done (3.468 secs)

🟩 Load5 < 0.5
🟨 0.5 <= Load5 < 1.0
🟥 Load5 >= 1.0

```
alias --> c14
Load5 = 0.7438658803
(Vanderbilt, Master, a19
alias --> c451
Load5 = 0.2021279732
(CHEP, Master, localho
alias --> c583
Load5 = 0.1111089841
(tam01.fnal.gov, Master
alias --> c350
Load5 = 0.8074037953
(citgrid3, Master, citgrid
alias --> c428
Load5 = 0.0311482463
(UW, Master, localhost)
alias --> c331
Load5 = 1.6379285638
(gocmon, Master, localh
alias --> c339
Load5 = 0.0887834982
(UC-Grid3, Master, loca
alias --> c412
Load5 = 0.1450199873
(UFL, Master, localhost)
alias --> c358
Load5 = 0.0408781920
```

PARAMETERS: [✓ Load5] Load15 Memory LostPackages Cpu_usr Active jobs Total jobs

HISTOGRAM: real-time [✓ hour] 12 hours 24 hours week month 3 months

Figure 5. *Full client*: only a selected part of the hierarchy is exposed

When a user accesses the same service using an iPAQ (Figure 6), the same data is represented using a combination of HTML pages (to specify the network configuration) and simpler 3D Cone trees with only subsets of nodes (e.g. a user chooses to view monitoring parameter values for a specific cluster). In cases where a large hierarchy needs to be presented, small display size and lower processing capabilities make it more efficient to present the data using a simpler method, such as a table that a user can scroll through.

The visualization is also displayed on the Sony Ericsson P800 mobile device, where data is again represented using a combination of HTML pages and 3D content (Figure 7).

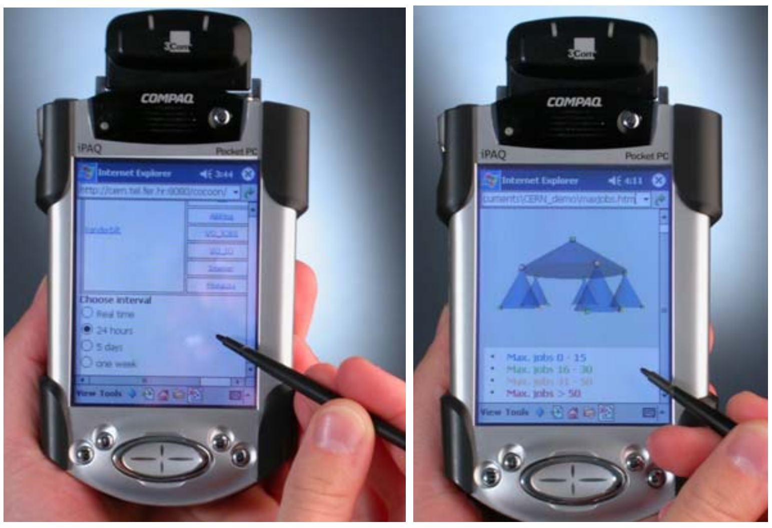


Figure 6. *Midi client (iPAQ)*: HTML and 3D content

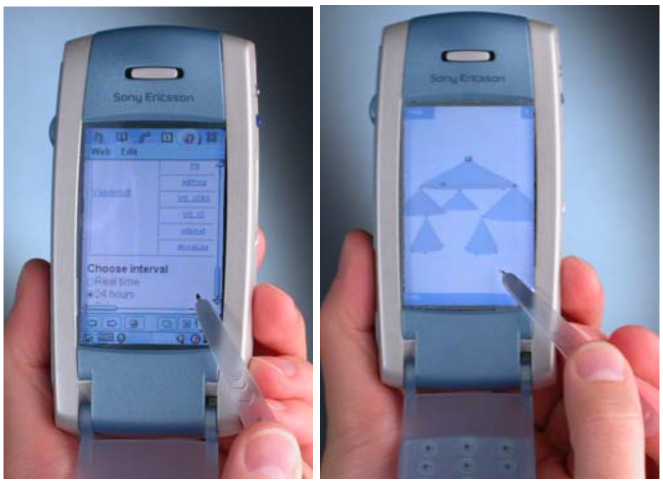


Figure 7. *Mini client (Sony Ericsson P800)*: HTML and 3D content

If a user using a WAP enabled mobile phone accesses the service, the result is data presented in WML format. Simple WML pages allow the user to list through the network configuration, and request monitoring parameter values (Figure 8).

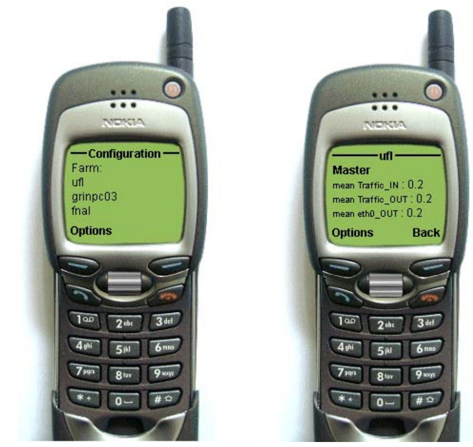


Figure 8. *Mini client: WAP enabled mobile phone*: WML content

5 Case study 2: Weather data on a geographical position

The second case study is a service named 3DWeather, which visualizes the current weather in Croatia, as reported by the Croatian Meteorological and Hydrological Service (DHMZ), using as representation small 3D objects, similar to TV forecast "icons" (cloud, arrow for wind direction, temperature scale), which are placed in the respective location on the map.

5.1 Implementation

The implementation of the MUVA architecture may again be discussed in terms its components: service logic, application interfaces, visualization tools, and platform drivers.

The users access the service via a standard Web interface, i.e. the Web browser. The client initiates the service by sending an HTTP request to the Java application servlet (*service logic*) located on the Web server. The servlet then launches the Java class (*application interface*), which retrieves meteorological data from the DHMZ Web page. The collected data includes the air temperature, atmosphere pressure, wind speed and direction, humidity, and overall weather conditions description (for example, sunny, cloudy, foggy, etc.). The retrieved data are stored in

XML format and the *visualization tools* Java classes are called to create a VRML scene. The XSLT transformations to transform the XML data to HTML tables have been implemented and used to create content for a mini client (WAP enabled mobile phone). For the full client display, data are represented by 3D objects, similar to TV forecast "icons" (cloud, arrow for wind direction, temperature scale), which are placed in the respective location on the map of the Republic of Croatia. Every 3D object has a built-in trigger, which, when clicked on, shows a window with exact data values. After the VRML scene is created the servlet also creates all the necessary HTML elements and returns the HTTP reply with the newly generated content. The client (Web browser) starts the Shout3D applet (*platform driver*) to present the returned VRML scene. The Shout3D applet enables the user to navigate through the VMRL scene.

The service may be accessed by using a full client (PC), midi client (iPAQ), and mini client (WAP enabled mobile phone). The Java servlet, which receives the HTTP request from the client, detects the user device and calls the *application interface* to retrieve the data. Once the data are retrieved and transformed to XML format, the *service logic* invokes the right *visualization tool* for the user device and the user's preferences, while later selecting the appropriate *platform driver.*

5.2 Results

A view of the full client display of the 3DWeather Web page is presented in Figure 9.

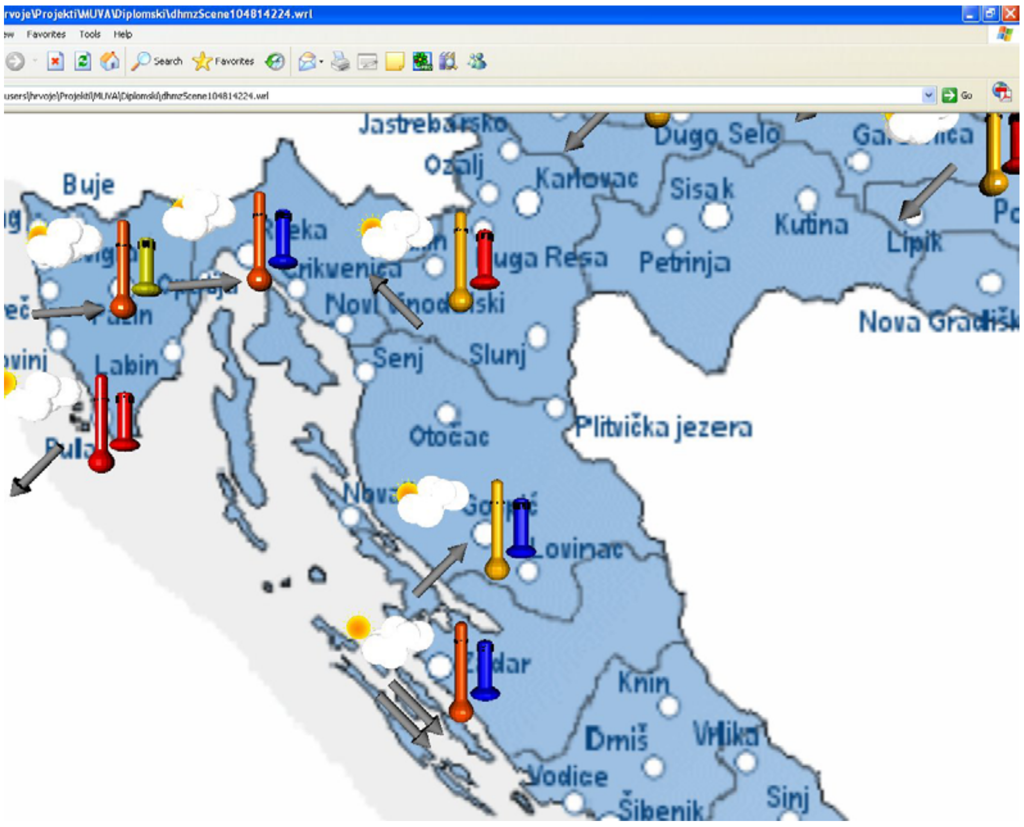


Figure 9. Full client display: Java applet showing the 2D map and 3D weather "icons"

A view of the midi client display of the 3DWeather Web page is presented in Figure 10.

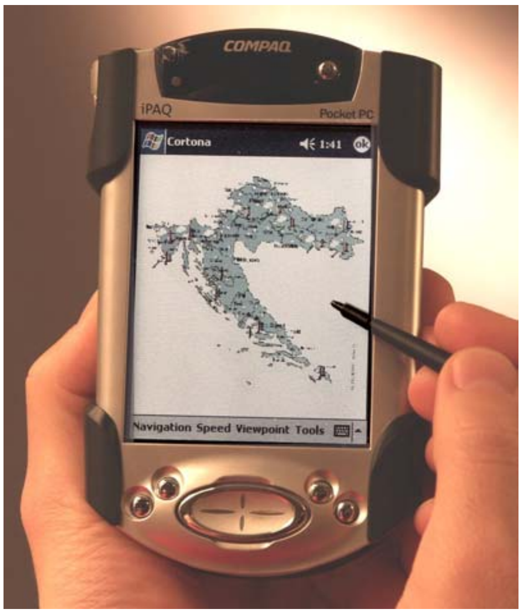


Figure 10. Midi client display: Java applet showing the 2D map and 3D weather "icons"

A view of the mini client display of the 3DWeather Web page is presented in Figure 11.

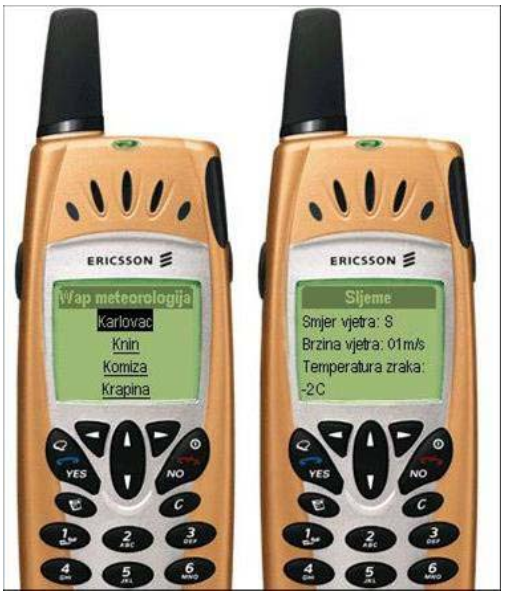


Figure 11. Mini client display: WML

6 Conclusions and future work

Most data visualization applications developed to date are designed to run on a target (or, a group of) client platforms, with limitations regarding new platforms and incorporation of new or various visualization techniques. In this paper, we have presented the concept of a Multiplatform Universal Visualization Architecture to address the issue of designing data visualization applications suitable for running on multiple platforms. The result is a modular, high-level, flexible visualization architecture that may easily be applied to various application domains. The key features are independence of data acquisition, as well as a thin adaptation "layer" for the platform on which the data is visualized.

The presented case studies demonstrate the visualization of Grid monitoring data and weather data on multiple platforms, with a separation of implemented software modules into four main MUVA components: visualization tools, platform drivers, application interfaces, and service logic. We discuss the limitations of various mobile platforms, and the need to adapt visualization techniques accordingly.

Further goals are to test the proposed architecture by focusing on new application domains, and possibly a greater variety of devices. This will help to determine the reusability of certain implemented modules, and open up areas and ideas for new visualization techniques.

7 References

[1] Information Technology – Computer graphics and image processing – The Virtual Reality Modeling Language (VRML) – Part 1: Functional specification and UTF-8 encoding. ISO/IEC 14772-1:1997

[2] BRITTON, K. H., CASE, R., CITRON, A., FLOYD, R., LI, Y., SEEKAMP, C. , TOPOL, B., AND TRACEY K. Transcoding: Extending e-business to new environments, *IBM Systems Journal 40*, 1, 153–178 (2001)

[3] BUTLER, M., GIANNETTI, F., GIMSON, R., AND WILEY, T. Device Independence and the Web. *IEEE Intenet Computing 6*, 5, 81–86 (2002)

[4] CARD, S.K., MACKINLAY, J.D., AND SHNEIDERMAN, B. *Readings in information visualization: using vision to think*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1999.

[5] COCKBURN A. AND MCKENZIE, B. An Evaluation of Cone Trees, In *People and Computers XIV: Proceedings of the HCI 2000*, 14th Annual Conference of the British Human Computer Interaction Group, 425–436 (2000)

[6] FOSTER, I., AND KESSELMAN, C. The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, San Francisco, CA, USA, 1998.

[7] HWANG, Y., KIM, J., AND SEO, E. Structure Aware Web Transcoding for Mobile Devices, *IEEE Internet Computing 7*, 5, 14–221 (2003)

[8] LAMBERTI, F., ZUNINO, C., SANNA, A., FIUME, A., AND MANIEZZO, M. An Accelerated Remote Graphics Architecture for PDAs. In *Proceedings Web3D 2003,* Saint Malo, France, 55–61 (2003)

[9] MEDVIDOVIC, N., MIKIC-RAKIC, M., MEHTA N. R., AND MALEK, S. Software Architectural Support for Handheld Computing, *IEEE Computer 36*, 9, 66–73 (2003).

[10] MUNOZ, M.A., RODRIGUEZ, M., FAVELA, J., MARTINEY-GARCIA, A.I., AND GONZALES, V. Context-Aware Mobile Communication in Hospitals, *IEEE Computer 36*, 9, 38–47 (2003)

[11] NEWMAN, H.B., LEGRAND, I.C., GALVEZ, P., VOICU, R., AND CIRSTOIU, C. MonALISA: A Distributed Monitoring Services Architecture. In *Proceedings of 2003 Conference for Computing in High Energy NuclearPhysics*, La Jolla, California, 2003.

[12] ROBERTSON , G.G., MACKINLAY, J.D., AND CARD, S.K. Cone trees: Animated 3D visualization of hierarchical information. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, ACM Press, New York, USA, 1991.