Elektrotehniški vestnik XX(Y): 1–6, YEAR
Electrotechnical Review, Ljubljana, Slovenija

Monitoring Data Visualization and Agent-based Software Management for the Grid

Maja Matijašević and Gordan Ježić

University of Zagreb, Faculty of Electrical Engineering and Computing, Unska 3, HR-10000 Zagreb, Croatia

Abstract. In this paper, we present our results in the areas of monitoring data visualization and software management for the Grid. We describe the concept of a Multiplatform Universal Visualization Architecture (MUVA) and its application to visualization of Grid monitoring data on multiple platforms. The key features of the proposed approach are independence of data acquisition and the thin adaptation "layer" for the platform on which the data is visualized. The presented case study demonstrates the visualization of Grid monitoring data on multiple client platforms. We also present a method, named Remote Maintenance Shell (RMS), developed for software management in large distributed environments. The proposed method is based on remote operations performed by cooperative mobile agents. A case study elaborates software migration and deployment in the Grid.

Key words: Grid, visualization, monitoring, mobile agent, software management

1 Introduction

Grid technologies have been described as supporting the sharing and coordinated use of diverse resources in distributed *virtual organizations* [6]. Due to its global size, as well as many geographically and organizationally distributed components, Grid is inherently difficult to monitor and manage. Tools and methods for addressing such tasks are needed, and they present an open area of research. In this paper, we describe our two contributions, one for visualizing Grid monitoring data, and the other for Grid software management. The paper is organized in two sections, addressing the respective contributions.

To date, a number of Grid monitoring systems has been developed to enable monitoring and displaying Grid topology, its components, and their parameters such as, for example, resource availability, load distribution, network input/output, and task performance [1][2][18]. In addition to data collection, most current systems also include a means for Web-based visualization of the monitoring information gathered by the system. With recent advances in multimedia and networking capabilities of user terminals, there is a strong trend towards visually more appealing and interactive user interfaces. Visualizations are becoming multi-modal and multi-platform, i.e. they may combine various media such as text, hypertext, pictures, multi-dimensional graphics, audio, and video, on a wide range of client (end-user) platforms, from PCs to new-generation mobile phones. To address the problem of data visualization on multiple client platforms, we propose the Multi-platform Universal Visualization Architecture (MUVA) [20], and apply it to Grid monitoring data. The data source used by our prototype was a central data repository provided by the MonALISA monitoring software (http://monalisa.cacr.caltech.edu/) [18], however, the described architecture is independent of data source, and could be tailored to work with a different database or data source by use of Web services. We also believe MUVA to be suitable for integrated monitoring systems [13], where data coming from different sources are presented through a common visual interface. The prototype service we implemented provides an interface for users to view Grid configuration and monitoring data, such as load, data rates, and memory on different platforms. In the first part of this paper we describe the MUVA architecture, and show its implementation and application onto Grid monitoring data visualization.

Software management, as one of the phases in the service provisioning process, involves operations such as software deployment, configuration, control, upgrading, provisioning, monitoring, accounting, billing, and self-management [14]. Desired capabilities of a software management system to be deployed in a large distributed environment, such as Grid, include [7][9][10][17]:

- management from a remote location
- management of multiple systems from a centralized location
- simultaneous management of multiple systems without handling each of them individually (easy configuration copying)

Received 26 December 2004
Accepted Day Month 2005

- software deployment to the remote system (migration and installation)
- software maintenance on the remote system (updating with a more recent version)
- software execution control (starting and stopping at will), and
- software tracing and testing on the actual target system.

To address the requirements of such a system, we propose an agent-based system, named Remote Maintenance Shell (RMS) [11][12]. RMS employs mobile software agents to perform Grid software management, while autonomously migrating from node to node during execution [5][3]. RMS provides the solutions for both basic (installation, starting and stopping) and advanced software management operations (testing and tracing), as well as performing them remotely. In the second part of the paper, we describe the RMS concept and implementation, and demonstrate its application in a case study based on the MonALISA monitoring software.

2 Multiplatform universal visualization architecture

The architecture we apply to visualization of Grid monitoring data is the Multiplatform Universal Visualization Architecture (MUVA) [20]. MUVA provides universal visual access to data independent of the client platform, while automatically adapting delivery modes to the particular platform. Fig. 1 presents the concept of the MUVA architecture.

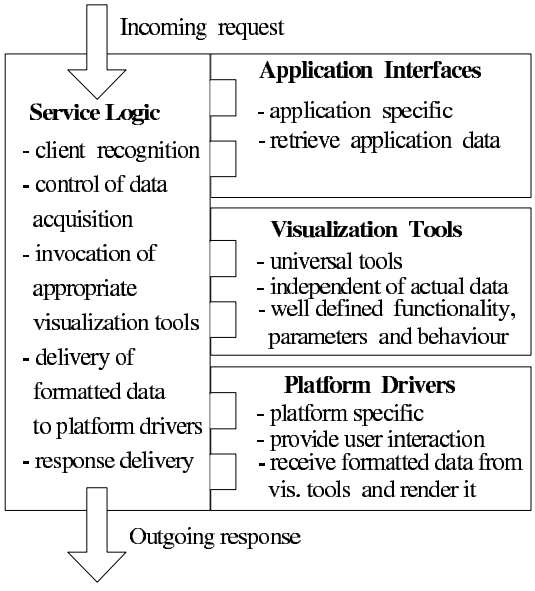


Figure 1. The concept of MUVA

MUVA comprises four components:

- *Visualization tools*, responsible for one particular mode of visualization (tool), e.g. a 3D structure of an input hierarchy, a list, a table, a bar-chart, a pie-chart, a tree, a graph, etc.
- *Platform drivers* for each supported platform, which render (visualize) formatted data received from visualization tools on the screen, and enable user interaction.
- *Application interfaces*, responsible for retrieving data from a data source via a standard application specific API, and converting data into Extensible Markup Language (XML) format.
- *Service logic*, which encompasses modules that provide the intelligence needed to connect components of the architecture in order to enable universal visual access and delivery mode adaptation.

MUVA has been designed as a flexible and modular architecture comprised of a collection of software modules. Key parts of the architecture are platform-independent "standardized" visualization tools, meaning that each tool has a predefined structure of data parameters and requests to be fed to it through its API. 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 utilized simply by developing thin application interfaces. The result is quick adaptability to various specific application domains. The service logic layer provides the necessary intelligence for coordinating application interfaces, visualization tools, and platform drivers, based on the identified client platform capabilities and user request.

To illustrate the typical usage scenario, we start from the incoming client request. Upon receiving a client service request, the client's preferences and platform capabilities are identified. One method of identification is based on W3C Composite Capabilities / Preference Profile (CC/PP) Recommendation for device independence [19], 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 using a URL. 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 are then sent to appropriate visualization tool(s). Formatted data received as the output from visualization tool are then delivered to the platform driver. The service logic layer provides the logic

necessary to select adequate visualization tools and platform drivers to produce the final content, adapted to the given client platform. In order to demonstrate the proposed approach in a real world scenario, a prototype Web based service was implemented, which provides multiplatform visual access to Grid monitoring data. 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.

2.1 Grid monitoring data visualization

The service implemented in this work provides an interface for users to view the logical structure of the Grid, combined with monitoring data, such as load, data rates, and memory on different platforms. In the Grid, the monitored nodes are arranged in a hierarchical manner into *farms* and *clusters*, referring to the geographical and/or logical grouping of *nodes* into virtual computing systems. As mentioned earlier, the data source used was a central data repository provided by the MonALISA system, which provides a distributed monitoring service and was in this case used to monitor hundreds of AliEn Grid sites (http://alien.cern.ch/). Users access the service by entering a unique URL, which is independent of the client 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 Virtual Reality Modeling Language (VRML), Hypertext Markup Language (HTML), and Wireless Markup Language (WML).

The service implementation was tested on several platforms: a desktop PC, a handheld PC, a Java-enabled PDA-type mobile phone, and a WAP-enabled mobile phone [20]. Where possible, the monitored network configuration (or a particular sub-configuration) was visualized using the 3D Cone Tree technique [19]. We consider applying other methods of visualization for other Grid properties as well [15]. Cone Tree is an interactive visualization technique suitable for hierarchical structures. The root of the network hierarchy is located at the tip of a 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. The user interface is enhanced by enabling interactive viewing, zooming, expanding and collapsing of parts of the structure.

The MUVA components are implemented as follows. Service logic is implemented using Apache Web server 2.0.47. and Apache Tomcat Server 4.1.27-LE-jdk14. Application interfaces use XML as an interchange format. The interface towards the actual data repository storing monitoring data collected by the MonALISA system is based on Web Services 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 to XML format. Visualization tools include various visualization techniques, including text, 2D graphics, and 3D graphics. Platform drivers were implemented for each platform (PC, handheld PC, and mobile phones with and without Java support). For 3D content, Shout3D applet and Cortona VRML plug-in were used for rendering on a PC, and Pocket Cortona ad Shout3d were used for rendering on the iPAQ PDA. For rendering on mobile phone with Java support (Sony Ericsson P800), we implemented a C++ application to dynamically generate a 3D scene by using the DieselEngine SDK 1.3, Symbian UIQ v7.0 SDK, and Metrowerks CodeWarrior for Symbian OS [16]. XSLT files were implemented to further adapt content for display on a particular device, including the creation of HTML and WML format for display on a WAP-enabled mobile phone. A view of the PC client display is shown in Fig. 2. Upon initial loading, the 3D view window (in

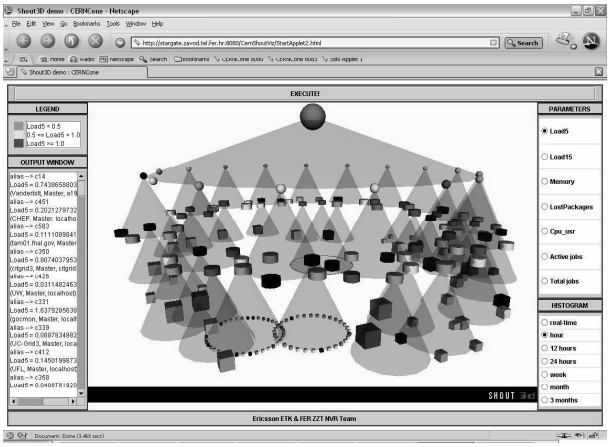


Figure 2. Visualization on a PC client

the middle) renders the 3D scene displaying the hierarchy of Grid farms, clusters and nodes. 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 (here: high, medium, low). For example, in Fig. 2 the selected parameter is one of those describing the load ("Load5"); so the 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. Fig. 3 shows a 3D view window and a tabular view on an IPAQ handheld PC. It may be noticed that the same data are now

Figure 3. Visualization on a handheld PC client

represented by using a combination of HTML pages and simpler 3D Cone trees with only subsets of nodes to adapt the view to the small display size and lower processing capabilities.

3 Remote Maintenance Shell

RMS is a distributed system comprising two main components: the client *RMS Console* and the server *RMS Maintenance Environment* (RMS ME), as shown in Fig. 4. The RMS Console offers a GUI through which the administrator may perform management actions on a remote system (RS). The computer on which RMS Console is installed and runs on will be referred to as the *management station*, as it represents the central location for performing the maintenance. The RMS ME, also referred to as *RMS Core*, is the server part of RMS and it has to be previously installed on managed RSs (in this case Grid nodes) in order for them to be managed by the RMS. All RMS operations are executed by mobile agents, which migrate autonomously between the network nodes during execution. Once the RMS user (i.e., the administrator of the managed system) defines operations to be performed at RS(s), the operations are assigned to one or more mobile agents, which then migrate to target RS(s) and perform the operations. We implemented both parts of RMS (the RMS ME and the RMS Console) in Java and we use Grasshopper (http://www.grasshopper.de) as the underlying agent platform.

The reason to apply mobile agents in RMS is to achieve decentralized operation execution and increased asynchrony of operation, as well as to reduce sensibility of RMS to network latency, and to facilitate flexible configuration of remote testing. In order for a specific software to be used with RMS, the software itself does not need to be modified; instead, only a thin adaptation layer has to be developed between the RMS and the software. This adaptation layer is called *software testbed*, and it is relatively simple to implement. The testbed is migrated to the remote systems together with the software, where it serves as an interface through which the RMS Core controls the software.

3.1 Agent Cooperation in RMS

In multi-agent systems, various organizational models may be used. In RMS, a hybrid between a master/slave and the agent teams models is used [8]. The RMS Console agent can be viewed as a master agent, because it intelligently decomposes a complex problem into a set of elementary tasks and distributes them to multi-operation agents. Agents communicate in order to execute all operations requested from them, and communication between agents follows the Foundation for Intelligent Physical Agents (FIPA) standards for inter-agent communication (http://www.fipa.org). In FIPA-compliant agent systems, agents communicate by sending messages. The message structure is written using Agent Communication Language (ACL). Message content is expressed in a content language, such as Knowledge Interchange Format (KIF) or Semantic Language (SL). The content expressions can be based on an ontology which defines the meaning of each message. In order to send an ACL message, it has to be encoded using the message representation appropriate for the transport e.g. XML, string or bit efficient representation. The transport message itself consists of an encoded ACL message plus the *envelope*. The envelope contains the sender and receiver transport descriptions which contain information on how to send the message. The envelope can also contain some additional information, such as the encoding representation, data related security, and other application-specific data

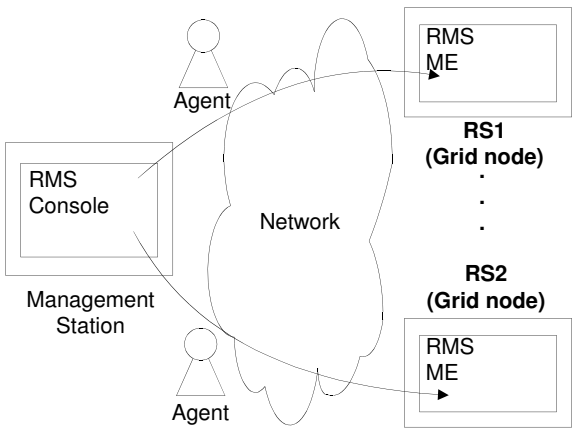


Figure 4. The concept of RMS

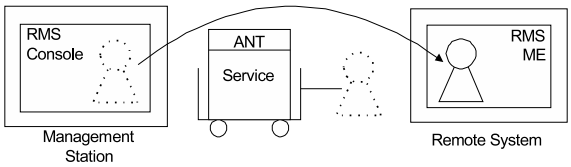
that must be visible for the transport or recipient(s). The transport message can be sent over the network by various protocols.

3.2 Grid software management by using RMS

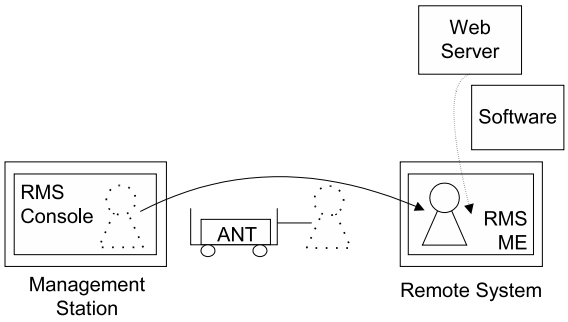
A case study presented here shows how RMS may be used to perform basic software operations in Grid environment, by installing and running the MonALISA software, a distributed monitoring software used in the real Grid environment. In order to install software on a given system, it is typically necessary to download the current release of software from the Web, unpack the archive file, and configure certain parameters on the target system. The goal of the demonstration is to install and run Grid software on three different remote systems.

First, in addition to the testbed, *installation archives* for both versions are created. The idea is that each software version must be packaged in an installation archive, which contains a special installation script. The installation archive will simply be referred to as "version". The installation script describes the configuration details that have to be changed when installing the software. When RMS agent executes the installation, it extracts the script from the archive, loads it with local specific parameters and initiates its execution. After that, the installation script takes over and does all the work. The parameters passed to the script include, for example, the local host name, the exact path that software will be installed to, etc. The format chosen for installation scripts is Apache Ant (http://ant.apache.org). Writing the installation script is typically the responsibility of the software testbed developer(s). Since the installation script describes the parameters to be configured when installing the software, its content can be easily extracted from the software documentation, which specifies those details for human users. In addition to specifying configuration details, Ant scripts can perfroming additional operations, such as downloading the software from a server in case that the software package to be migrated is too large to be efficiently transported by the agent itself. In this case, the agents transfer only the Ant script, and during installation, the script downloads the actual software from a HTTP server to the remote system. Both of these approaches are depicted in Fig. 5.

At the beginning of the scenario, RMS ME is started on all three remote systems (RS1, RS2, RS3), none of which initially contains the MonALISA software. The RMS user starts the RMS Console and performs registration at all three remote systems. In the first part of the scenario, the RMS user wants to install the MonAlisa software on all three remote systems. To achieve that, one has to pick out a Java Archive (JAR) file containing the testbed and a JAR file representing the installation archive for that version, and set the software to be started on one



(a) software carried by the agent



(b) software downloaded from the server

Figure 5. Alternative software migration scenarios

of the remote systems. After that, the configuration may easily be copied to other systems. Once the user initiates the task execution, four agents are created; namely, Agent1, Agent2, Agent3, and, Agent4. Agent1 migrates the software to all three remote systems (RS1, RS2, RS3), by performing the following operations on each of them: testbed migration, version migration. Agent2 migrates to RS1, and - after having received the notification from Agent1 that the software has been migrated to that location - executes the following operations: testbed installation, version installation, setting execution parameters, starting the software. Agent3 migrates to RS2 and performs the same tasks there, while Agent4 does the same at RS3. Version installation includes unpacking of the installation archive and executing the Ant script provided in the archive. After the execution has been completed, the MonALISA software may be observed as up and running on all three remote systems.

4 Conclusions

In this paper, we describe our results in the area of visualization of monitoring data and remote software management for the Grid. We have presented MUVA and its application to visualization of Grid monitoring data on multiple platforms. The key features of the proposed approach are independence of data acquisition and the thin adaptation "layer" for the platform on which the data are visualized. The presented case study demonstrates the visualization of Grid monitoring data on multiple platforms. We have also presented the RMS system and its application to Grid software management. The case study demonstrates the installation of MonALISA monitoring software on three Grid nodes. Ongoing work on RMS includes introducing additional advanced software manage-

ment features, such as centralized trace collection from multiple systems. We also plan to enable intelligent software management by using semantic agents.

Acknowledgement

This work has been supported by Ericsson Nikola Tesla R&D Centre under research projects “Networked Virtual Reality support in IMS” and “Remote Operation Management and Multiagent System”, and the Ministry of Science, Education and Sports of the Republic of Croatia, under project 0036030 “Mobility of Users and Services in New Generation Networks”.

5 References

[1] S. Andreozzi, N. De Bortoli, S. Fantinel, A. Ghiselli, G. Tortone, C. Vistoli, GridICE: a monitoring service for the Grid, *Proceedings of the Third Cracow Grid Workshop*, Cracow, Poland, 2003, pp. 220-226.

[2] Z. Balaton, P. Kacsuk, N. Podhorszki, F. Vajda, Comparison of Representative Grid Monitoring Tools, *Computer and Automation Research Institute of the Hungarian Academy of Sciences*, Tech. Report LPDS-2/2000, 2000. [Available: http://www.lpds.sztaki.hu/publications/reports/lpds-2- 2000.pdf]

[3] F. Bergenti, M-P. Gleizes, F. Zambonelli, *Methodologies and Software Engineering for Agent Systems*, Kluwer Academic Publishers, 2004.

[4] M. Butler, F. Giannetti, R. Gimson, T. Wiley, Device Independence and the Web, *IEEE Internet Computing*, vol. 6, no. 5, 2002, pp. 81-86.

[5] W. R. Cockayne, M. Zyda, *Mobile Agents*, Prentice Hall, 1997.

[6] I. Foster, C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 1988.

[7] I. Foster, C. Keselman, J. Nick, S. Tuecke, Grid Services for Distributed System Integration, *IEEE Computer*, vol. 35, no. 6, 2002, pp. 37-46.

[8] G. Ježić, M. Kušek, S. Dešić, A. Carić, D. Huljenić, Multi-Agent System for Remote Software Operation, *Lecture Notes in Artificial Intelligence*, LNAI 2774, Springer-Verlag, 2003, pp. 675-682.

[9] G. Ježić, M. Kušek, T. Marenić, I. Ljubi, I. Lovrek, S. Dešić, B. Dellas, Grid Service Management by Using Remote Maintenance Shell, *Lecture Notes in Computer Science, LNCS 3270*, 2004, pp. 136-150.

[10] G. Ježić, M. Kušek, I. Ljubi, T. Marenić, I. Lovrek, S. Dešić, B. Dellas, Mobile-Agent Based Software Management in Grid, *Proc. of the Workshop on Emerging Technologies for Next Generation GRID*, Modena, Italy, 2004, pp. 345-346.

[11] M. Kušek, G. Ježić, I. Ljubi, K. Mlinaric, I. Lovrek, S. Dešić, O. Labor, A. Carić, D. Huljenić, Mobile Agent Based Software Operation and Maintenance, *Proceedings of the 7th International Conference on Telecommunications ConTEL 2003*, Zagreb, 2003, pp. 601-608.

[12] G. Ježić, M. Kušek, I. Lovrek, S. Dešić, B. Dellas, Agent-based Framework for Distributed Service Management, *Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems*, Boston, MA, USA, 2004, pp. 583-588.

[13] M. Mambelli, R. Gardner. Integration of Monitoring Systems for Grid Environments, *Proceedings of the 13th IEEE International Workshop on Enabling Technologies (WET-ICE): Infrastructure for Collaborative Enterprises*, Modena, Italy, 2004, pp. 266-267.

[14] T. Marenić, G. Ježić, M. Kušek, I. Lovrek, Using Remote Maintenance Shell for Service Provisioning in the Distributed Systems, *Proc. of the International Telecomunications Symposium VITEL 2004*, Maribor, Slovenia, 2004.

[15] D. Mikić, L. Skorin-Kapov, O. Dobrijević, P. Schilhard, 3D Visualization of the Geographical & Organizational Structure of the Grid, *Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference MELECON 2004*, Vol. II, Dubrovnik, Croatia, 2004, pp. 689-692.

[16] M. Mošmondor, H. Komerički, I. Pandžić, 3D Visualization of Data on Mobile Devices, *Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference MELECON 2004*, Vol. II, Dubrovnik, Croatia, 2004, pp. 645-648.

[17] J. Nabrzysku, J. M. Schopf, and J. Weglarz, *Grid Resource Management - State of the Art and Future Trends*, Kluwer Academic Publishers, 2004.

[18] H. B. Newman, I. C. Legrand, P. Galvez, R. Voicu, C. Cirstoiu, MonALISA: A Distributed Monitoring Service Architecture, *Proceedings of the 2003 Conference for Computing in High Energy Nuclear Physics*, La Jolla, California, USA, 2003, MOET001, 8 pp.

[19] G. G. Robertson, J. D. Mackinlay, S. K. Card. Cone trees: Animated 3D visualization of hierarchical information, *Proceedings of the SIGCHI Conference on Human factors in computing systems: Reaching through technology*, ACM Press, New York, USA, 1991, pp. 189-194.

[20] L. Skorin-Kapov, D. Mikić, H. Komerički, M. Matijašević, I. Pandžić, Multiplatform Universal Visualization Architecture, *Proceedings of the Second International Conference on Advances in Mobile Multimedia 2004*, Bali, Indonesia, 2004, pp. 15-24.

Maja Matijašević received her Dipl.-Ing. (1990), M.Sc. (1994), and Ph.D. (1998) degrees in Electrical Engineering from the University of Zagreb, Croatia, and the M.Sc. in Computer Engineering (1997) from the University of Louisiana at Lafayette, LA, USA. She is presently an Assistant Professor in the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. Her main research interests include networked virtual environments and advanced multimedia services for next generation networks. Dr. Matijašević is a member of IEEE, ACM, and Upsilon Pi Epsilon Honor Society in the Computing Sciences.

Gordan Ježić received his Dipl. Ing., M.Sc. and Ph.D. degrees in Electrical Engineering at the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia, in 1995, 1999 and 2003, respectively. He currently works as an Assistant Professor at the Department of Telecommunications of the Faculty of Electrical Engineering and Computing, University of Zagreb. His research interests include mobile process modeling, formal methods, distributed computing, agent-oriented software engineering, mobile software agents, and multi-agent systems. Dr. Ježić is a member of IEEE.