Towards a Universal Client for Grid Monitoring Systems: Design and Implementation of the Ovid Browser

Marios D. Dikaiakos, Artemakis Artemiou and George Tsouloupas

Dept. of Computer Science
University of Cyprus
Nicosia, CY-1678, Cyprus
{mdd,cs01aa2,georget}@cs.ucy.ac.cy

Abstract

In this paper, we present the design and implementation of Ovid, a browser for Grid-related information. The key goal of Ovid is to support the seamless navigation of users in the Grid information space. Key aspects of Ovid are: (i) A set of navigational primitives, which are designed to cope with problems such as network disorientation and information overloading; (ii) A small set of Ovid views, which present the end-user with high-level, visual abstractions of Grid information; these abstractions correspond to simple models that capture essential aspects of a Grid infrastructure. (iii) Support for embedding and implementing hyperlinks that connect related entities represented within different information views; (iv) A plug-in mechanism, which enables the seamless integration with Ovid of third-party software that retrieves and displays data from various Grid information sources, and (v) a modular software design, which allows the easy integration of different visualization algorithms that support the graphical representation of large amounts of Grid-related information in the context of Ovid's views.

1 Introduction

Large Grid infrastructures typically host a plethora of monitoring tools and services, which produce, collect, maintain, and publish data about the configuration, the structure, and the state of Grid resources [10, 15]. The collection of all that data constitutes a large, complex, dynamic, and fragmented information space, which we call the *Grid information space*. Seamless access to this space is invaluable for Grid end-users, application developers, and administrators, as it can help them gain a better understanding of Grid configuration, operation, and state, providing useful insights into properties otherwise obscured by Grid virtualization. Nevertheless, in order to access Grid-related information, Grid users have to use different types of monitoring client software. The choice of the monitoring client depends on the type of information sought, the Grid monitoring system used to collect that information, and the middleware deployed on the Grid infrastructure of interest. Different monitoring clients, however, employ different protocols to retrieve information from monitoring sources, support different human-computer interaction paradigms, have customized user-interfaces, and display information mostly in "raw" formats, such as listings or tables. Recent Grid monitoring systems export their information in HTML, XML, or graphical formats, thus making it accessible through Web-based front ends [3, 6, 14]. Nevertheless, Web-based presentations of monitoring data are organized also as collections of listings and tables that provide only a partial view of a Grid infrastructure, as captured by the sensors of the corresponding monitoring systems. Thus, the adoption of the Web browser as a common client-side system for delivering monitoring information relieves end-users from the need to employ cumbersome command-line instructions or special-purpose tools; however, it does not support the view of a coherent information space, inside which an end-user can seamlessly navigate and retrieve Grid-related information. Consequently, discovery and retrieval of information about the status and configuration of Grid infrastructures remains a daunting experience and a major obstacle to the Grid's wider adoption.

The goal of our work is to design a "universal" monitoring-client software that will replace the plethora of existing and emerging monitoring clients,

1-4244-0054-6/06/$20.00

while implementing functionalities that will enable Grid users to: (i) retrieve and visualize information from different monitoring systems; (ii) navigate seamlessly into the Grid information space, discovering dynamically and accessing available Grid information sources; (iii) maintain a coherent view of the Grid information space by placing retrieved monitoring information into proper context. To meet this goal, we designed and implemented *Ovid*, a browser for Grid-related information. The design and implementation of Ovid addresses the following key challenges: (i) In contrast to the Web and to traditional hypertext systems, Grid-information retrieval cannot be accomplished with a single protocol or function call. Grid information sources are heterogeneous and comply to different communication and query protocols (LDAP, HTTP, SOAP, proprietary). Therefore, we need a monitoring-client that can retrieve information from different sources, using different protocols on the back-end, while it maintains the view of a coherent information space on the end-user side (see Figure 1). (ii) Grid-related information is not organized in hypertext form: information retrieved from Grid information sources does not contain explicitly encoded links between related information items. Therefore, the implementation of navigational functionalities requires the definition and management of hyperlinks inside the monitoring client. (iii) The structure and presentation of information retrieved from Grid information sources is "source-oriented," in the sense that it reflects the functionality and data organization of each particular source. Consequently, in order to establish the view of a coherent information space, the monitoring-client has to provide and support a visual representation of retrieved information that is close to the user's perception of the Grid.

In this paper, we present the design and implementation of Ovid and discuss how Ovid addresses the challenges mentioned above. The remaining of the paper is organized as follows: Section II provides a presentation of the functionality currently supported by Ovid. Section III describes the software design of Ovid. Sections IV and V discuss in more detail how Ovid supports navigation and plug-ins. We conclude in Section VI.

2 Key Concepts

2.1 Overview

Ovid is designed to support end-user navigation inside a virtual information hyperspace, whose structure is defined by a model of the Grid architecture. Hyperspace nodes correspond to entities of the Grid model.

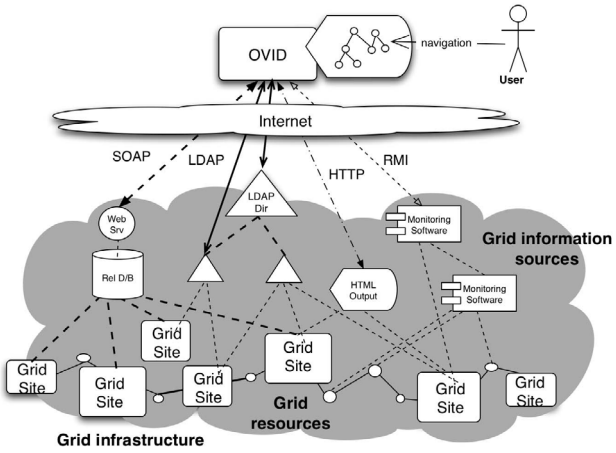

Figure 1. The concept of Ovid.

Node content is retrieved dynamically from Grid information sources providing metadata about real resources that correspond to the model entities. Hyperlinks that connect hyperspace nodes correspond to relations derived from the model's structure. Key features of the Ovid browser are:

- A set of primitives, which are commonly found in Web browsers and hypertext systems, and have been proven essential in supporting end-user navigation inside large information spaces and in coping with problems such as network disorientation and information overloading: `hyperlinks`, `back` and `next` keys, `bookmarking` and `history` mechanisms, and `text-search` panels [13]. These primitives help end-users navigate inside the Grid information space using a simple hypertext-like interaction paradigm.

- A graphical representation of a Grid infrastructure model capturing key aspects of Grids. This model is used to organize internally the information retrieved from various sources and to help the user establish the mental concept of a coherent information space. Ovid embeds hyperlinks in the graphical representation of the Grid model, turning it into a spatial hypertext map.

- A *plug-in mechanism*, which enables the dynamic integration in Ovid of third-party monitoring clients that retrieve and display data from disparate Grid information sources.

The combination of simple navigation primitives with the plug-in mechanism and the hypertext-map encoding of the Grid, hides the complexity of protocols and

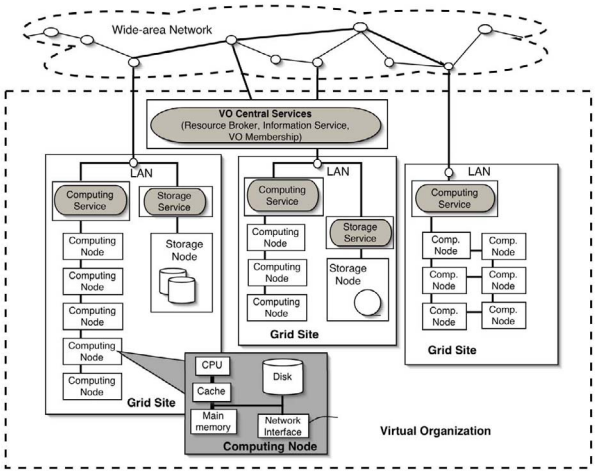

Figure 2. A model of a Grid infrastructure.

systems that are used to retrieve Grid-related information. Furthermore, the navigation functionality supported by the GUI of Ovid enables users to maintain a view of the Grid as a unified information space, where it is relatively easy to locate useful information through simple navigation and search mechanisms (see Figure 1).

2.2 Ovid Views: a graphical representation of a Grid model

Grid infrastructures consist of geographically distributed *Grid Sites*, which belong to different administrative domains, communicate through Internet, and make their resources available to one or more *Virtual Organizations* [1, 8]. Typically, a *Grid site* comprises a cluster of computing and storage nodes interconnected through a high-speed local-area network. Each Grid site also hosts a number of *local services*, which manage the site's membership to a Grid infrastructure by providing remote access to local resources and information. The operation of Grid infrastructures is supported by *central services*, which publish information about the configuration and state of resources, handle authentication and security issues, and facilitate job submission and control.

This view of a Grid infrastructure (depicted in Figure 2), represents an architectural abstraction of several large Grid testbeds, such as EGEE [1] and is a high-level model for the Grid. Basic entities of this model are: (i) Abstractions of Grid resources, such as sites, clusters, storage devices, worker nodes (computing hosts), and network links; these abstractions are represented as rectangular boxes and links in Figure 2; (ii) Abstractions for Grid services, which are represented as oval shaped boxes in Figure 2; services manage access control to resources and publish information about resource configuration and status; for instance, *Computing Elements* manage access to computational resources of Grid Sites and *Storage Elements* act as interfaces to local storage devices.

The model of Figure 2 captures the basic configuration of a Grid infrastructure as perceived by its end-users and can be easily translated into an extensible ontology for Grids, organizing and placing into context Grid-related information. It is worth noting that this model reflects in part the structure of the *GLUE Information Model* (also known as GLUE Schema), which is a set of information specifications capturing key aspects of Grid architecture [4]; GLUE is used as the data model of choice for the Information Services of several large testbeds.

Ovid presents graphically different aspects of the model of Figure 2 in order to support end-user navigation inside the Grid information space and to convey the notion of a coherent Grid information hyperspace to its end-users. To this end, the browser displays in its main panel a set of *Ovid Views*, which represent an instantiation of the model for some specific infrastructure of interest. Currently supported Ovid Views represent graphically the following entities:

- The collection of Grid sites that participate to a selected Virtual Organization (see Figure 3).
- The collection of resources that belong to individual Grid sites (see Figure 4).
- The network topology interconnecting these sites (see Figure 6).
- Simple information derived from Grid Information Services about Grid sites, their resources, their status, etc.

Ovid Views are *spatial hypertext maps*: they store attributes of the Grid entities they represent, such as identifier (URI), type, and status; also, they contain statically embedded hyperlinks, which encode hierarchical containment or reference relationships between the interlinked entities of the Grid model. Ovid supports also the dynamic installation and invocation of external hyperlinks through its Views; these hyperlinks point to information that is retrieved from plug-ins of third-party monitoring services and placed into proper context.

2.3 Software design

Ovid's software is based on the object-oriented *Model-View-Controller* (MVC) design paradigm [12]. This paradigm divides the functionality of an object-oriented application into three categories: the *Model*, the *View* and the *Controller*. The Model category contains the data source in which all data manipulation and processing operations take place. The View category contains all the "views" derived by the corresponding model. Typically, these views are visual or textual representations of manipulated data. The Controller category handles user interaction with the software and transforms user request into messages, which are sent to the Model in order to produce the desired view with the requested information. In our implementation, we adopted the *Model-Delegate* variation of MVC, which was introduced by Sun Microsystems in its Java Swing GUI components [9]. This variation merges the View and Controller categories into one category named *Delegate*.

The use of the Model-Delegate architecture in Ovid is very important: each Grid information source is managed by a different Model-Delegate module. Each Model component encapsulates the queries that can be issued upon its corresponding information source, speaks the source's communication protocol, manages the transactions between the source and the browser, receives and interprets the source's output, casting it into a format tailored to Ovid's internal data structures.

2.4 Ovid functionality

Ovid is invoked as a stand-alone client program on a Grid-user's workstation; its user interface is similar to that of known Web browsers and consists of a *navigation toolbar*, an *input bar* for inserting information through the keyboard, and a *main display panel*, which is used to display available Ovid Views.

Navigation and Search functionality: Ovid provides several options to facilitate end-user navigation through the Grid information space. A user can start his navigation by typing-in the domain name of a central Information Server of a Virtual Organization at the input-bar of Ovid; this opens the *VO-Sites View* to the main display panel (see Figure 3). The VO-Sites View represents all the sites (clusters) that belong to the selected Virtual Organization. This representation displays each site as a star-shaped collection of resources, with the *Computing Element* located at the center of the star, connecting together its associated *Worker Nodes* and *Storage Elements*. Different colors

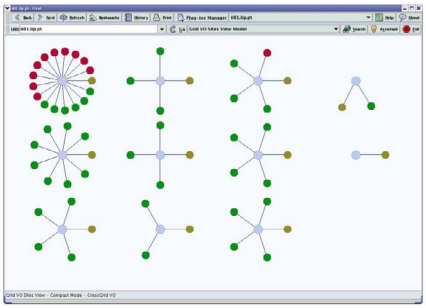

Figure 3. VO-Sites View: Grid sites of the CrossGrid testbed [2]

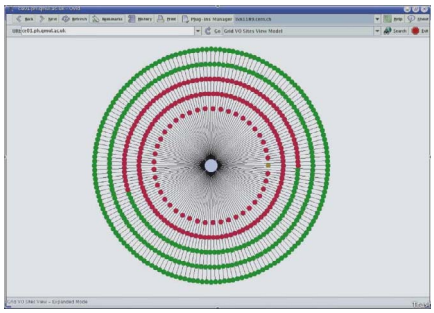

Figure 4. Expanded-Site View.

are used to represent the state of sites and resources (accessible, free, busy).

The VO-Sites View is a hypertext map to the information space of the selected VO. For instance, when the user clicks at the center of a chosen site, Ovid displays that site's visual representation expanded in a new panel (see Figure 4). Furthermore, Ovid implements a "mouse-over" functionality on top of this expanded view: when a user rolls his mouse over the visual representation of some resource, a tool-tip appears providing textual information about attributes of the actual resource (its name, its local resource manager, etc). The user can click on the visual representation of the resource in order to retrieve and display more information about it.

Users can select the `Back` and `Next` buttons, to move back and forth in their navigation history. The `History` mechanism allows users to view their recent navigation history and jump directly to previous navigation steps by clicking on some history item. Ovid supports also the bookmarking of Ovid Views: users can select the `Bookmarks` button to retrieve a small graphical-dialog window that allows them to store their current View as a bookmark or to reactivate a stored

Ovid View. Finally, Ovid provides a `Refresh` button for retrieving and displaying up-to-date information on the current Ovid View.

Ovid users can perform keyword-based searching for Grid resources: by selecting Ovid's `Search` button and typing-in a keyword, users can get a list of all the available resources with a domain name that matches the given keyword. Selecting an item from this list brings the corresponding resource's display in Ovid's main panel.

Plug-in functionality: An important property of Ovid is its support for installing and invoking external plug-ins. This support is provided through the *Plug-in Manager* of Ovid. External plug-ins are components that retrieve and display information from Grid information sources (e.g., monitoring systems) attached to Grid resources. End-users can use this manager to discover and activate supported plug-ins, and to extend the information that can be displayed through Ovid, dynamically. To this end, users can right-click on a Grid-resource representation to get a pop-up menu that displays all information sources attached to those resources, and to invoke the corresponding plug-ins for retrieving and displaying monitoring information.

3 System Design

The system design of Ovid comprises three main parts: (i) a set of modules and data-structures that manage the *internal state* of Ovid and end-user location inside the virtual Grid hyperspace; (ii) a set of *front-end* modules that manage Ovid's graphical-user interface and end-user interaction support, and (iii) a set of *back-end* modules handling the interaction with various Grid information sources. The front and back-end modules are grouped together in model-delegate components. An overview of the design is presented in Figure 5.

The **Shared Data Structures** of Ovid are used to organize and store information reflecting the structure of the Grid model described in Figure 2. For instance, information maintained in the Shared Data Structures represents the sites that comprise a VO, the resources that belong to a site, the monitoring services supported by a specific resource, the network topology of a VO, etc. The modules of Ovid make use of these data structures to support virtual-navigation in the Grid information space.

The virtual "location" of a user during his navigation inside the Grid information hyperspace is represented by a **Context** object. This object has the following attributes: (i) the active Virtual Organization; (ii) the Grid resource that the user has selected

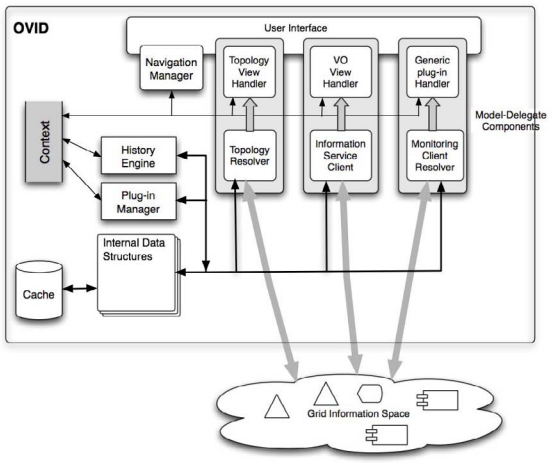

Figure 5. Ovid's design diagram.

in his last navigation step (a Grid site, node, etc); (iii) the type of the selected resource; (iv) the "Model-Delegate" entity that is active and provides information about the selected resource, and (v) the user security certificate (if available). Ovid uses the Context object in order to interpret end-user input (mouse clicks and keyboard input) properly. Whenever an end-user interacts with Ovid's GUI (e.g., to navigate to a new location), the Context is updated with information reflecting the user's new virtual location and the new state of Ovid.

Changes in the Context are monitored by the **History Engine** module, which stores Context objects to the hard disk in order to maintain a log of successive Contexts. The History Engine uses this log to support the history-dependent navigation mechanisms provided with buttons, such as `Back` and `Next`.

The **Plug-in Manager** provides support for plug-in installation, removal, and modification. Whenever a plug-in is installed, the Manager records in the Shared Data Structures of Ovid configuration information representing the name of the plug-in, the protocol it uses to retrieve data from its associated Grid information service, and the types of resources that support interaction with that plug-in.

Ovid's front-end comprises a **User Interface** module, which manages the generic interaction functionality provided by the browser through its navigation and input toolbars. Navigation operations are handled by the **Navigation Manager** and can be invoked when the user presses the `Back`, `Next`, and `Refresh` buttons, or types inside the `Search` input bar and hits return. Whenever any of these navigation primitives is acti-

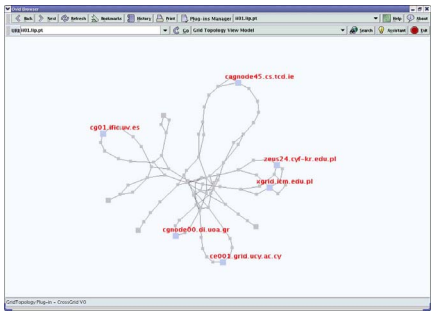

Figure 6. VO-Topology view.

vated, the Navigation Manager invokes other modules to update Ovid's internal state and display.

The main display panel of the browser is controlled by Model-Delegate components, which create and manage the various Ovid Views. Currently, Ovid incorporates two default Model-Delegate components: VO-Sites and Network Topology. Additional components can be invoked dynamically through the plug-in mechanism described later. The **VO-Sites** module consists of the *Grid Information Service (GIS) client* Model and the *VO View Handler* Delegate. The GIS client queries the information service of a selected Virtual Organization to retrieve information regarding the sites of that VO and their properties (names of Computing and Storage Elements, numbers and names of constituent Worker Nodes, status information, etc). This information is stored in the Shared Data Structures of Ovid and used by the VO View Handler for creating the VO visual representation (see Figure 3). Besides visualizing the VO structure, the VO View Handler takes control of the interactions that the end-user has with the VO View. In order to overcome problems that arise from network disconnections, the data retrieved from the GIS client are kept in the cache of Ovid.

The **Network Topology** module consists of the *Topology Resolver* Model and the *Topology View Handler* Delegate. The Topology Resolver discovers the underlying network topology of a selected Virtual Organization. To this end, it uses the Grid to execute recursive, distributed, pairwise traceroutes between all sites that belong to the selected VO. Alternatively, it can retrieve this information from monitoring tools such as TopoMon [7]. The collected traceroute information is stored in a traceroute log file kept in Ovid's cache. This file is used by the *Topology View Handler*, which implements a graph-visualization algorithm to display the VO-Topology view (see Figure 6), and handles the user interaction with that particular view.

Other Model-Delegate components can be added dynamically to Ovid as plug-ins. **Plug-ins** are supported by the Ovid architecture in order to facilitate the extension of the browser's functionality with new features, new views, and new types of information. In a plug-in component, the Model module acts as a software client to some Grid monitoring system or information service, whereas the Delegate module provides a visual representation of the information retrieved from the Model and supports end-user interaction. Notably, one Model can be associated with multiple Delegates to provide for alternative visual representations and interaction paradigms with the same information source.

4 Implementation Issues

Navigation support: As mentioned earlier, Ovid supports a set of generic navigation primitives commonly found in most Web browsers. These primitives are chosen in order to help users build and maintain a mental model of their virtual navigation path in the Grid information space, thus facilitating the location of useful information and the avoidance of network disorientation problems [13]. Access to Ovid's navigation primitives is given via `Back, Next, Refresh, Stop, History` and `Bookmarks` buttons, and the input bar where the user can type-in or paste plain text. All these user-interface entities are placed at the navigation toolbar of Ovid.

When the `Back` or `Next` primitives are invoked, the Navigation Manager of Ovid exchanges messages with the History engine to gain access to Ovid's history file and retrieve information pointing to the Context that should be restored. If the context found belongs to an Ovid's default Model, the visualization information of the restored context can be retrieved from the cache. If the restored context belongs to an external plug-in, Ovid directly passes a call to the context's corresponding Model in order to update the active display panel of Ovid with the visualization information of the restored Context.

When the `Refresh` primitive is invoked, the Navigation Manager reads Ovid's active state from the Context, asks the active Model to retrieve fresh data from its associated Grid information source, and invokes the active Delegate module to redraw its visual representation of the retrieved data. The `Search` primitive is invoked when a user types a search string at the input bar of the search dialog. In that case, the Navigation Manager retrieves a list of entities that can be matched against the given string, and compares them to the string. The results of this matching are presented to the user. If the user clicks upon a matched entity, the Navigation Manager invokes the proper Model-

Delegate where the corresponding resource is registered to, in order to produce a view of that resource.

The `History` button invokes the *History* window, which provides access to history data maintained by the History Engine. The History window supports the direct selection and display of a stored view or the removal of entries from the history logs. Similarly, the `Bookmarks` button invokes a small window that allows the user to store the active state of Ovid, in order to easily access it in the future.

Hyperlinks: The navigation functionality of Ovid is supported by hyperlinks. A hyperlink is a clickable object-geometry embedded in an Ovid View and associated with a *hyperlink resolver*. The resolver is a piece of code that is part of the View's Delegate module and is activated by a click on its object-geometry. Upon activation, the hyperlink resolver determines the semantics of the click and identifies the Model that needs to be invoked in its response. To this end, it inspects the current Context object and retrieves information regarding the selected resource and its associated Models. Then, the resolver makes a call to the appropriate Model, passing as argument the current Context and information about the resource selected with the click. The Model replies by instructing its Delegate to update the active View.

In our current implementation, whenever a user clicks on a Grid resource in the VO-Sites view, the hyperlink resolver gets from Context the domain name and type of the selected resource. Then, it makes the appropriate call to the Information Service Client, which turns the control of Ovid's main panel to its View module, providing the corresponding view.

Plug-ins: One of the principal advantages of Ovid is that it can retrieve and display information from a variety of Grid information sources, without having to "hard-wire" the code that handles information retrieval and visualization into its implementation. The concept of plug-ins is central to the achievement of this characteristic.

An Ovid plug-in is a small, pluggable, Model-Delegate-based component that implements the `ConnectionClass` interface specified inside Ovid. `ConncectionClass` specifies methods such as: (i) `getMenu`; this returns a menu-list with all the operations supported by the plug-in; (ii) a `listener` method, which handles user selection of the plug-in menu's choices; (iii) `refreshModel`, which instructs the plug-in's Model module to update the information displayed by its Delegate module; (iv) `handleURISelection`, which takes as a parameter the domain name of a resource and retrieves information about it from the Model, and

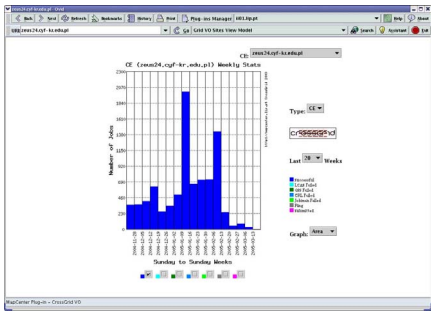

Figure 7. Using the MapCenter plug-in.

(v) `giveSupportedResourceTypes`, which returns a vector containing the resource types supported by the plug-in.

In the current implementation of Ovid we have implemented plug-ins for the MapCenter [6] monitoring system (see Figure 7), which runs on several large testbeds, such as EGEE, DataGrid and CrossGrid, and for the JIMS monitoring system [5], which runs on the CrossGrid testbed. Furthermore, the VO-Sites and the VO-Topology modules are implemented as plug-ins, which are integrated by default in Ovid and support its generic navigational functionality.

The registration and management of plug-ins is handled by the Plug-in Manager (Figure 5). End-users can interact with this manager through a small GUI component, which displays a dialog window providing access to all operations that can be executed on the plug-ins. If the user selects the "install" operation, he is presented with a list of available plug-ins. After selecting a plug-in for installation, the Plug-in Manager updates the configuration information it maintains and proceeds automatically with the registration of Grid resource types supported by the plug-in. To this end, it queries the plug-in to retrieve this information. The GUI displays also a list with all plug-ins already installed in the browser. The user can choose a plug-in from the list, in order to update it or remove it.

5 Conclusions and future work

Navigation inside large information spaces is a key challenge in modern, open, large-scale, distributed systems. This challenge arises in the context of the Grid, since its dynamic nature, heterogeneity, scale, and open architecture make the use of modeling abstractions for Grid infrastructures an essential condition for harnessing their power. Such abstractions must hide the numerous details involved in Grid infrastructures, while maintaining and providing end-users and administra-

tors with high-level "views" of important aspects of Grids, such as the capacity of resources available in a Virtual Organization, the measured performance of Grid nodes and VOs, the average network distance and bandwidth between two nodes, and the network topology linking the nodes of a VO.

To address these challenges, we designed and developed Ovid, a tool for modeling Grid infrastructures visually, and for navigating through these visual representations using the familiar interaction paradigm of Web browsers. Ovid has an open and flexible architecture that enables the easy integration of different visualization models and algorithms. Also, it supports the easy installation and invocation of plug-ins, enhancing the visualization of properties derived from a variety of monitoring systems and Grid services. Currently, we are extending Ovid by developing plug-ins for different monitoring systems and generic Web services. Furthermore, we are investigating the development of better algorithms for visualizing large-scale Grid infrastructures with hundreds of Sites and thousands of nodes. Finally, we are investigating approaches for generalizing the management of Ovid's implicit hyperlinks, based on approaches like the ones described in the Dexter hypermedia model [11].

Acknowledgments: This work was supported in part by the European Union under the CrossGrid project (contract number IST-2001-32243).

References

[1] EGEE: Enabling Grids for eScience in Europe. http://www.eu-egee.org, (accessed April 2004).

[2] European CrossGrid Project. http://www.crossgrid.org (accessed April 2005).

[3] S. Andreozzi, N. De Bortoli, S. Fantinel, A. Ghiselli, G. Tortone, and C. Vistoli. GridICE: A Monitoring Service for the Grid. In *Proceedings of the Third Cracow Grid Workshop*, pages 220–226, October 2003.

[4] S. Andreozzi et al. GLUE Schema Specification, version 1.2. http://infnforge.cnaf.infn.it/projects/glueinfomodel/ (accessed Apr. 2005).

[5] Kazimierz Balos and Krzysztof Zielinski. JIMS - the Uniform Approach to Grid Infrastructure and Application Monitoring. In *4th Cracow Grid Workshop 2004*, December 2004.

[6] F. Bonnassieux, R. Harakaly, and P. Primet. MapCenter: an Open GRID Status Visualization Tool. In *Proceedings of ISCA 15th International Conference on parallel and distributed computing systems*, September 2002.

[7] M. den Burger, T. Kielmann, and H.E. Bal. TopoMon: A Monitoring Tool for Grid Network Topology. In Sloot, Tan, Dongarra, and Hoekstra, editors, *Proceedings of the International Conference on Computational Science-Part II*, volume 2331 of *Lecture Notes in Computer Science*, pages 558–567. Springer, 2002.

[8] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3):200–222, 2001.

[9] A. Fowler. A Swing Architecture Overview. The inside Story on JFC Component Design. http://java.sun.com/products/jfc/tsc/articles/architecture/ (accessed Nov. 2004).

[10] M. Gerndt, R. Wismueller, Z. Balaton, G. Gombas, P. Kacsuk, Z. Nemeth, N. Podhorszki, H-L. Truong, T. Fahringer, M. Bubak, E. Laure, and T. Margalef. Performance Tools for the Grid: State of the Art and Future. APART White Paper. Technical Report Research Report Series, vol. 30, University of Technology Munich, SHAKER Verlag, 2004.

[11] Frank G. Halasz and Mayer D. Schwartz. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, 1994.

[12] G. E. Krasner and S. T. Pope. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object Oriented Programming*, 1(3):26–49, August/September 1988.

[13] J. Nielsen. *Multimedia and Hypertext: The Internet and Beyond*. Morgan Kaufmann, 1995.

[14] R. Wolski, L.J. Miller, G. Obertelli, and M. Swany. Performance information services for computational grids. In Jarek Nabrzyski, editor, *Grid Resource Management, State of the Art and Future Trends*, pages 193–213. Kluwer Academic Publishers, 2003.

[15] S. Zanikolas and R. Sakellariou. A Taxonomy of Grid Monitoring Services. *Future Generation Computer Systems*, 21(1):163–188, January 2005.