A Component-based Framework for Uniform Resource Visualization

Kukjin Lee
Department of Computer Science & Engineering
Michigan State University
East Lansing, MI 48824
Phone:(517) 355-9974

Email: leekukji@cse.msu.edu

Diane T. Rover
Department of Electrical & Computer Engineering
Michigan State University
East Lansing, MI 48824
Phone: (517) 353-7735
Fax: (517) 353-1980
Email: rover@egr.msu.edu

Keywords: resource monitoring, performance visualization, component-based development, visualization composition, uniform representation

1 Introduction

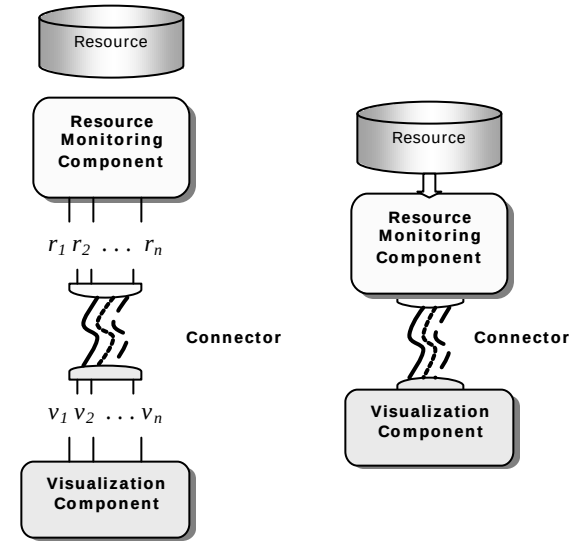
The PGRT project has prototyped a framework for integrating performance analysis tools for parallel and distributed systems and for visualizing system and application performance [7]. One result of PGRT is a component-based specification language called the Visual Object Markup Language (VOML) [4][5][6]. VOML is an SGML-based language for specifying on-line performance visualization components. The framework includes two key features that support an integrated environment for performance problem solving: (1) portability across platform-dependent user interface toolkits, and (2) a flexible component-based visualization architecture, EPIRA (Event Processing and Information Rendering Architecture). Furthermore, the framework is object-oriented and easily distributable via middleware software such as CORBA and DCOM. VOML is based on two visual-object levels: a device-dependent low level, and a device-independent high-level. VOML uses SGML for structuring visual objects, and Scheme language scripts for defining performance visualization semantics. The use of SGML enables development of a performance visualization infrastructure, from which a designer may construct platform- and tool-independent visual objects. It may also facilitate automatic monitoring, analysis, and visualization of wide-area distributed applications via network-enabled SGML entity managers. The use of Scheme for visual object semantics enables both rapid prototyping of visual objects and customizing of VOs for a wide range of platforms.

The Uniform Resource Visualization (URV) project, leveraging PGRT outcomes, has introduced a new paradigm for constructing visualizations and for monitoring and viewing complex parallel/distributed systems [8]. URV supports: (1) standard visualization services; (2) creation of system-level views through composition; and (3) sharing of visualization design knowledge. Uniformity in URV is implemented by describing resource and visualization components with XML-based formal descriptors. This facility helps developers to design uniform interfaces for accessing, viewing, and managing heterogeneous resources. Composition refers to support for multi-resource visualization. With the composition facility, the designer can construct a computational system with multiple resources and view system-level performance across resources. Reusability in URV is provided via descriptor registry and discovery. Using a directory service, users can retrieve existing visualizations, and by composing them, can construct a new visualization. Distinctive services of URV include: (1) metadata service to define visualizations for any resource in the system, ranging from hardware devices to application software modules; (2) connection service to bind resource(s) to a visualization; and (3)

composition service to construct a new visualization based on multi-level, composable, reusable, and distributed components.

2 Uniform Resource Visualization

A visualization in URV, termed a *URV view*, is associated with one or more resources, where a resource may be a physical entity (e.g., router, sensor) or a logical entity (e.g., process, scheduler). It consists of a resource part and a visualization part, as illustrated in Figure 1. The resource part of a URV view consists of a *resource* and a *resource-monitoring component.* A resource-monitoring component provides performance monitoring and control services; it separates a visualization from the physical or logical resource being visualized. The visualization part of a URV view is termed a *visualization component.* Each component specifies services comprised in its interface to other components, e.g., $r_1\ r_2 \ldots\ r_n$ and $v_1\ v_2\ \ldots v_n$ in Figure 1(a). An interface, called a *connector*, defines a set of services that are provided by one component and required by another component.



(a): Elements of URV (b): URV view

Figure 1. URV view and its elements

A connector binds services between components. However, compared with binding the names of services in sequential components, plugging together components in a distributed system requires an explicit description [2][3]. For example, the mechanism for forwarding data between components may be designed as a unit-sized buffer or a variable-length buffer. A variable-length buffer ensures that all measured data are eventually visualized. Alternatively, a unit-sized buffer that discards all but the most recent value prevents the visualization component from lagging behind the resource-monitoring component. POLYLITH [3] supports these alternatives by encapsulation in separate *software busses*, whereby a designer can choose an appropriate bus to connect the invocation of a required service in one component to the definition of the service provided by another component. URV supports similar flexibility through its connection service.

3 Component-based Framework for URV

We are developing a component-based framework for structured development of URV views, as shown in Figure 2. A URV framework consists of metadata (*visualization descriptor*), a metadata processing service (*metadata service*), three URV management services (*directory service, connection service, composition service*), and two repositories (*component repository, connector repository*).

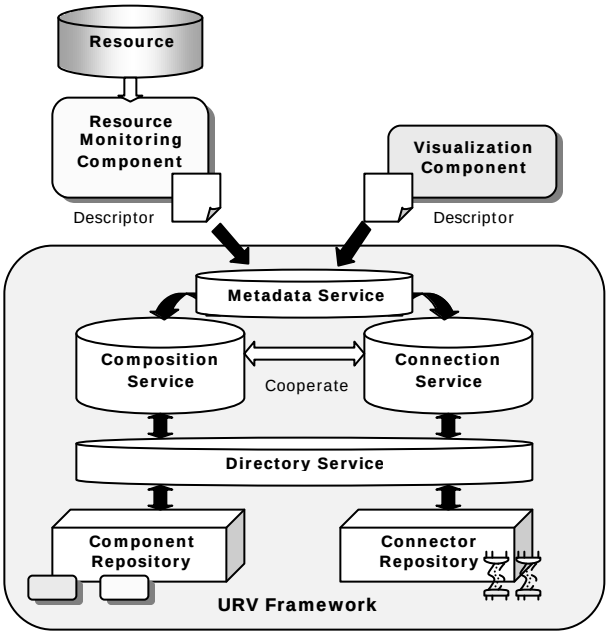


Figure 2. URV framework

A descriptor provides a representation of a component at a level of detail necessary for its invocation and interconnection, hiding other details. This facility helps developers design uniform interfaces for accessing, viewing, and interacting with heterogeneous resources. A metadata service provides descriptor processing methods such as parsing. A directory service enables users to register resource-monitoring/visualization components for retrieval by other users. Using a directory service, the framework can locate qualifying components at local or remote sites. A connection service maintains executable connectors between resource-monitoring and visualization components, and provides methods for configuring and controlling a connector. Connectors are run-time entities that implement a set of rules describing how connected components interact and react, e.g., to transfer performance data. A composition service supports the creation of a new visualization from defined components based on a set of rules.

3.1 Visualization Descriptors

Descriptors, in general, allow a designer to specify the interfaces to components rigorously and to specify component interconnection at these interfaces. An ideal description of a component encompasses the component's *concept, content*, and *context* [9]. Concept is a description of what the component does, including its interface and any operating specifications. Content describes how the concept is realized or implemented so that users of the component can modify or adapt it to a specific use. Finally, context describes the domain of applicability of the component. In a visualization descriptor, the attributes of data model and interface describe the component's

concept, and the attributes of graphical representation, its content. This content is essential to rules for visualization composition, which is based on shared attributes of graphical representations [1]. The attributes of component identification (e.g., name, location, execution environment) in a visualization descriptor denote the component's context. A URV directory service uses this context to locate and invoke a particular component. These descriptions are coded in XML, with advantages such as readability and availability. With this approach, components may be specified uniformly and be reusable and interoperable.

3.2 Connection Service

A URV connection service provides executable connectors between resource-monitoring and visualization components, as well as control of these connectors. Each connector is uniquely typed to provide specific interface characteristics. The connector must also handle heterogeneity, such as the components themselves or their interaction. For example, depending on the synchronization policy, a connector must adjust the service invocation timing and data transferring protocols. Meanwhile, designing every single connector for all different components is almost impossible. Instead, we are developing a small set of template connectors. Each template connector is configurable to support families of related components that share the same interface and the same attributes of graphical representation. Thus, connector design addresses a range of configurations and operations.

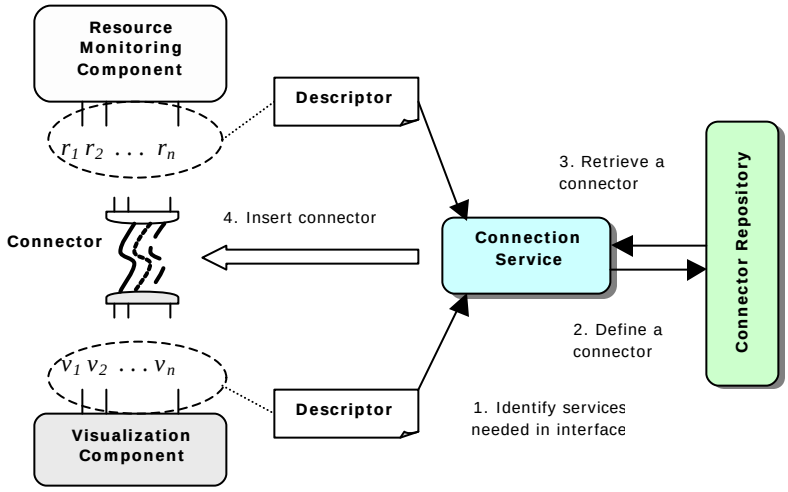


Figure 3. URV connection service

Figure 3 depicts the basic operations of a connection service and their sequencing. First, a connection service identifies resource-monitoring and visualization components with the necessary contexts. Next, referencing the component's concept, the connection service evaluates the feasibility of the connection by comparing the interfaces of the components. If it is a feasible connection, the connection service chooses a template connector that supports these interfaces. Then the connector is configured; it is specialized by binding the names of services. Finally, the connection service plugs components together by invoking the connector. Once invoked, the connector controls the interaction between components.

3.3 Composition Service

URV can be used to systematize the visualization of resources. Moreover, the uniformity, or consistency, in this systematic approach can be leveraged to help designers compose URV views. Composition enables the automated presentation of system-level, multiple-resource views in a dynamic, heterogeneous environment. It also simplifies the process of view creation for typical users, not unlike the use of a chart wizard in a desktop spreadsheet tool. Composing visualizations—either by merging visualization layers or by designing a new visualization layer—is difficult. Figure 4 depicts the basic process of visualization composition by a composition service. Each component is represented by a descriptor, where *r* denotes a resource-

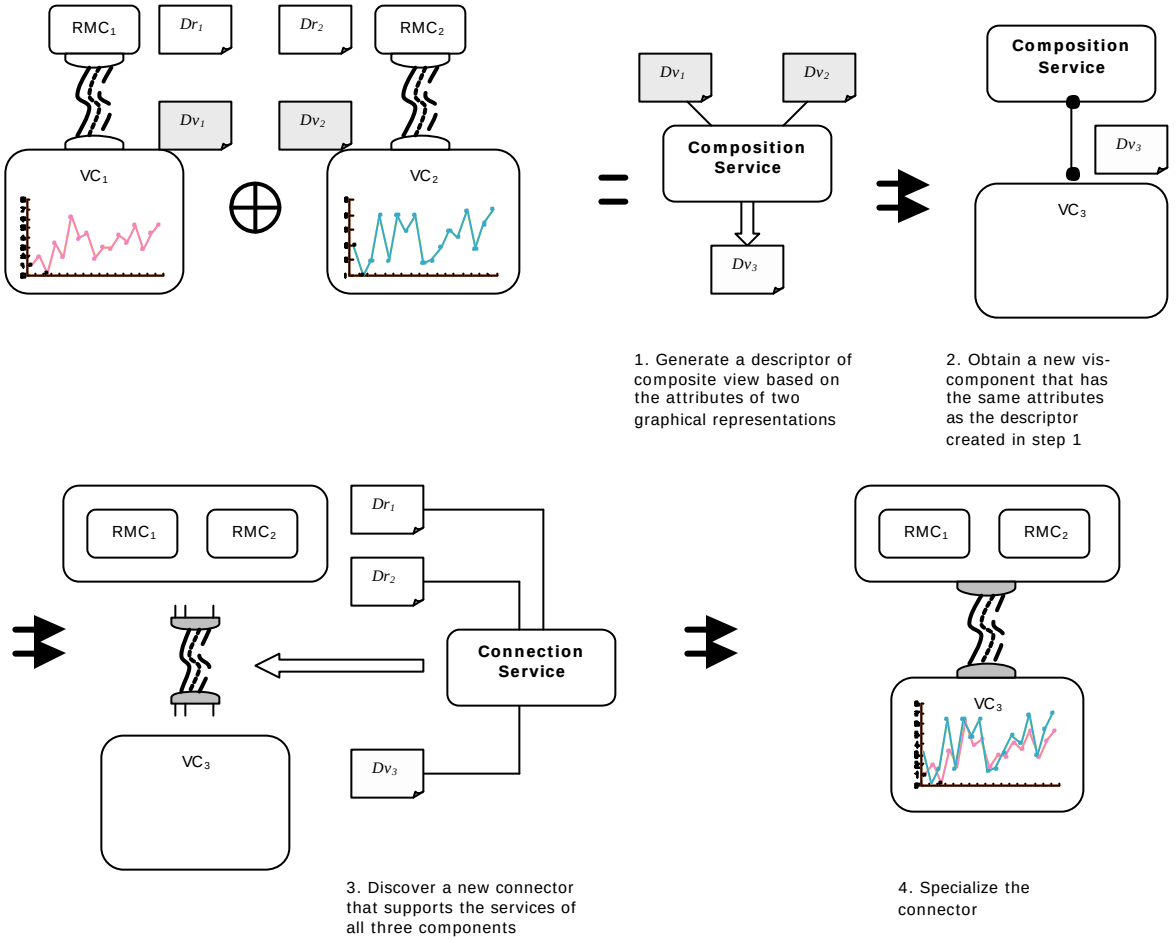


Figure 4. Basic process of visualization composition

monitoring component (RMC), and *v*, a visualization component (VC): Dr_1 ,Dr_2 ,Dv_1 ,Dv_2 ,*and* Dv_3. The type of composition shown in Figure 4, indicated by the ? operator, is referred to as *union*. It results in a new URV view that synchronizes/joins (spatially and temporally) individual views into a single graph based on a common independent variable. Synchronization composition between two URV views depends on a concrete set of transformation rules; it is straightforward given design knowledge about the visualization components. In the example shown, we can specify a *design transformation* rigorously that matches two different line graphs sharing a common independent variable and replaces these graphs with a single graph.

We are developing a database of such *composition transforms*, each of which matches multiple URV view descriptions and replaces them with a synchronized, but otherwise functionally equivalent, view. The process by which the transformations are applied is also depicted in Figure 4. Given two URV views, the separate instances of visualization descriptors, Dv_1 and Dv_2, are first aggregated and then replaced through a transformation by a new specification, Dv_3. Then, the composition service obtains a new visualization component (VC_3) that supports the same interface specified in Dv_3. Based on all three descriptors, Dr_1, Dr_2, *and* Dv_3 , the connection service selects a new connector so that the new visualization directly interacts with both resource-monitoring components, RMC_1 and RMC_2. Finally, the connector is specialized by binding the services of the resource-monitoring components (RMC_1 and RMC_2) and the visualization component (VC_3).

URV can also aid in the design of new visualization layers that represent the cooperation between/among resources. This type of composition is referred to as *synthesis*. It is a conscious activity by a designer that cannot be automated as fully as union. It requires indexing of visualization components according to a classification scheme. To replace an aggregate of two independent views with a new one, the designer *decouples* the aggregate visualization from the resource-monitoring components (aggregate resource) and specifies a set of search attributes to guide the search for a new visualization component. A new URV view is completed by connecting services in the aggregate resource to services in the visualization and vice versa.

Acknowledgements

This work was supported in part by NSF grant ACI-9624149 and uses results of DARPA contract DABT63-95-C-0072. The URV concept was developed in collaboration with Michigan State University professors Matt Mutka and Kurt Stirewalt.

References

[1] J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Transactions on Graphics*, 5, 1986, pp. 110–141.

[2] R. Allen and D. Garlan, "A formal basis for architectural connection," *ACM Transactions on Software Engineering and Methodology*, 6(3), July 1997, pp. 213-248.

[3] J. M. Purtillo, "The polylith software bus," *ACM Transactions on Programming Languages and Systems*, 16(1), January 1994, pp. 151-174.

[4] Aleksandar Bakic, M. Mutka, and D. Rover, "VOML Reference Manual," Michigan State University, July 1998. Corresponds to documented software package available in public domain at http://www.egr.msu.edu/Pgrt

[5] Aleksandar Bakic, M. Mutka, and D. Rover, "VOML Tutorial," Michigan State University, July 1998.

[6] Aleksandar Bakic, Matt W. Mutka, and Diane T. Rover, "An on- line performance visualization technology," In *Proceedings of the IEEE Heterogeneous Computing Workshop*, April 1999, pp. 47–59.

[7] Diane T. Rover, Abdul Waheed, Matt W. Mutka, and Aleksandar Bakic, "Software tools for complex distributed systems: Toward integrated tool environments," *IEEE Concurrency*, 6(2), April-June 1998, pp. 40–54.

[8] Kuk-jin Lee and D. T. Rover, "A Component-based Framework for Uniform Resource Visualization," technical report, Michigan State University, 2001.

[9] W. Tracz, "Where does reuse start?," In *Proc. Realities of Reuse Workshop*, 1990.