


2006

Uniform resource visualization

Kukjin Lee
Iowa State University

Follow this and additional works at: <https://lib.dr.iastate.edu/rtd>

 Part of the [Computer Sciences Commons](#), and the [Electrical and Electronics Commons](#)

Recommended Citation

Lee, Kukjin, "Uniform resource visualization " (2006). *Retrospective Theses and Dissertations*. 1271.
<https://lib.dr.iastate.edu/rtd/1271>

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

Uniform resource visualization

by

Kukjin Lee

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:

Diane T. Rover (Major Professor)

Dan Berleant

Yong Guan

Suraj Kothari

G. M. Prabhu

Iowa State University

Ames, Iowa

2006

Copyright © Kukjin Lee, 2006. All rights reserved.

UMI Number: 3217285

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 3217285

Copyright 2006 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Graduate College
Iowa State University

This is to certify that the doctoral dissertation of
Kukjin Lee
has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

DEDICATION

To my beloved wife and parents

TABLE OF CONTENTS

LIST OF FIGURES	vii
LIST OF TABLES	x
ACKNOWLEDGEMENTS	xi
ABSTRACT.....	xii
CHAPTER 1. INTRODUCTION.....	1
1.1 Performance Visualization	1
1.2 Motivation	3
1.3 Problem Statement	5
1.4 Objectives.....	9
1.5 Contributions.....	10
1.6 Organization of Dissertation	12
CHAPTER 2. VISUALIZATION MODEL	13
2.1 View	13
2.2 Uniform Resource Visualization (URV).....	15
2.3 Performance Visualization in URV.....	16
2.4 Summary	20
CHAPTER 3. PERFORMANCE VISUALIZATION KNOWLEDGE.....	21
3.1 Performance Visualization Knowledge (PVK).....	21
3.2 Performance Visualization Ontology (PVO)	23
3.3 PVO Representation.....	24
3.3.1 <i>Class and Class Hierarchy</i>	24
3.3.2 <i>Properties</i>	26
3.3.3 <i>Instances</i>	28
3.4 Ontology Query.....	32
3.5 Summary	33

CHAPTER 4. VISUALIZATION COMPONENT	36
4.1 Visualization Component (VC).....	36
4.2 Methodology for Creating Visualization Component.....	38
4.2.1 Step 1: Define Visualization Knowledge	40
4.2.2 Step 2: Design a Visual Design.....	42
4.2.3 Step 3: Determine Graphical Primitive.....	43
4.2.4 Step 4: Implement Graphical Primitives	45
4.2.5 Step 5: Implement a Template Visualization Component.....	51
4.2.6 Step 6: Implement a Visualization Component.....	54
4.3 Summary	56
CHAPTER 5. VISUALIZATION COMPOSITION	57
5.1 Visualization Composition.....	57
5.2 Methodology for View-Level Composition (Synthesis).....	62
5.2.1 Step 1: Compose Knowledge	62
5.2.2 Step 2: Identify New Visualization Component(s).....	64
5.2.3 Step 3: Construct a New Visualization Component (VC).....	65
5.3 Summary	67
CHAPTER 6. SPREADSHEET FOR SYSTEM-LEVEL	
PERFORMANCE VISUALIZATION	68
6.1 Spreadsheet Visualization	68
6.2 Spreadsheet for System-Level Performance Visualization	69
6.3 Summary	73
CHAPTER 7. PUTTING IT ALL TOGETHER	74
7.1 URV System.....	74
7.1.1 URV Services.....	76

7.2 Case Study	78
7.2.1 <i>Performance Scenario</i>	78
7.2.2 <i>Testbed</i>	80
7.2.3 <i>Problem Diagnosis Scenario</i>	81
7.3 Summary	90
CHAPTER 8. RELATED WORK	92
8.1 Visualization Taxonomy	92
8.2 Automatic Visualization.....	93
8.3 Component-based Visualization	95
8.4 Performance Visualization	97
8.5 Performance Monitoring Tools	98
CHAPTER 9. CONCLUSIONS.....	102
9.1 Summary of Contributions	102
9.2 Future Work	105
APPENDIX. PERFORMANCE VISUALIZATION ONTOLOGY	108
BIBLIOGRAPHY	121

LIST OF FIGURES

Figure 1. Netlogger Visualization [76] supports system-level visualization which is only based on temporal-based synchronization. It allows add more data, but the view type is fixed.....	8
Figure 2. View and Visual Elements	14
Figure 3. Performance visualization with/without URV	16
Figure 5. GMA and URV Implementation	20
Figure 6. DAML representation of a class (Performance data) and sub class (Performance Metric Data), and the example of BarChart visual design class	26
Figure 7. Visual Design class and Its properties.....	27
Figure 8. DAML representation of supportAnalysisType Property	28
Figure 9. RDF Triple model for What Supports Interaction Analysis Type	32
Figure 10. DAML representation of “What Visual Designs Support Interaction Analysis Type”.....	34
Figure 12. Visualization Component.....	37
Figure 13. Knowledge of VC.....	41
Figure 14. Elements of Bar Chart visual design	43
Figure 15. Class hierarchy of graphical primitive	46
Figure 16. Graphical primitive class definition	48
Figure 18. Class hierarchy of visualization component.....	51
Figure 19. Visualization component class definition.....	53
Figure 20. Visualization component interfaces	54
Figure 21. Pseudo code of Visualization Component (VC)	55
Figure 22. Bar Chart VC creation.....	55

Figure 23. Component-level visualization composition	58
Figure 24. Graphical primitive level visualization composition.....	60
Figure 25. View-level visualization composition	61
Figure 27. <i>VC1</i> (displays the occurrences of send events with markers over time)	63
Figure 28 <i>VC2</i> (displays the occurrences of receive events with markers over time).....	64
Figure 29. <i>VC3</i> (displays the status of processes with different colors (yellow: idle, green: busy)).....	64
Figure 30. A RDF-based query model of the composed knowledge.....	65
Figure 31. A new composite VC.....	66
Figure 32. System-Level Performance Visualization (SSPV).....	71
Figure 33. RMC directory browser.....	72
Figure 34. Matched VC and its preview dialog.....	72
Figure 35. URV-based performance visualization system	75
Figure 36. Query service.....	77
Figure 37. Performance scenario testbed.....	81
Figure 38. Select a RMC that provides the process status information	83
Figure 39. Matched Gantt Chart component for process status.....	83
Figure 40. Gantt Chart components added to the worksheet.....	84
Figure 41. Merge two process status VCs by merge-union	85
Figure 42. Visualization of Send/Receive events	86
Figure 43. Space diagram visualization component	87
Figure 44. Visualization of CPU load information.....	88
Figure 45. Composed space-diagram with CPU load information.....	88
Figure 46. Observing correlation by aligning two VCs vertically on SSPV	89
Figure 47. Snapshot of Snap-Together [77]: Snaps multiple visual interfaces	

(e.g., tree view, table view) to explore file hierarchies.....97

Figure 48. Extension of SSPV107

LIST OF TABLES

Table 1. Cluster and Grid Performance Visualization (Representative Subset).....	2
Table 2. Instances of Classes of Analysis, Resources, and Performance Data.....	28
Table 3. Selected Instances of Visual Design Class	31
Table 4. Effectiveness rankings of different perceptual encoding techniques for three basic data types (Adapted from [70]).....	44
Table 5. Preferred data type for each graphical primitive and its attribute.....	44
Table 6. Effectiveness rankings of graphical primitive for each data type.....	45

ACKNOWLEDGEMENTS

First of all, I would like to thank Dr. Diane Rover. Dr. Rover has been a great advisor and significant influence in my life. I have learned so many things, including how to conduct a research, how to write a paper, and the technical contents of performance visualization. Her constant support and encouragement have been the most important stimulus for me to get back on track when I feel frustration and hopefulness. I am truly proud of being her student. Also, I thank my dissertation committee members, Dr. Daniel Berleant, Dr. Yong Guan, Dr. Suraj Kothari, and Dr. Gupur Prabhu, for dedicated times for review and many suggestions for improvements.

At last, I thank my family for consistent support. Most of all, I thank my beloved wife, Seunghye Jang. Without her, simply this work would not exist. Her beautiful mind and crystal clear soul have inspired me all the time. Thank you, sweetheart. I adore you.

ABSTRACT

Computing environments continue to increase in scale, heterogeneity, and hierarchy, with resource usage varying dynamically during program execution. Computational and data grids and distributed collaboration environments are examples. To understand performance and gain insights into developing applications that efficiently use system resources, performance visualization has proven useful. Performance visualization tools, however, often are specific to a particular resource at a certain level of the system, possibly with fixed views. Thus, they limit a user's ability to observe a performance problem associated with multiple resources across system levels and platforms. To address this limitation, information integration is necessary. In this research, we propose a new performance visualization framework, Uniform Resource Visualization (URV), focusing on integration of performance information into system-level visualizations. The goal of URV research is to systemize the performance visualization of resources with reusable and composable visualizations.

Keywords: performance visualization, visualization composition, knowledge representation, component development, grid systems

CHAPTER 1. INTRODUCTION

This Chapter describes basic information on performance visualization, including a selected list of well-known performance visualizations. Then, we discuss the motivation, objectives, and contributions of the research,

1.1 Performance Visualization

A synopsis of software visualization for parallel/distributed computing is found in an encyclopedia article by Rover [95]. One type of software visualization is program visualization. Program structure and behavior may be elucidated through graphical representation, allowing programmers to diagnose problems from a high to low level [95]. In addition to program visualization, performance visualization has extended and enhanced users' understanding of the mapping of a program onto a computer. A comprehensive article by Heath, Malony, and Rover [52] presented examples and concepts of the visual display of performance data. Performance visualization technologies/tools (e.g., [83][17][85]) support the development of visualizations. Unfortunately, rapid and structured development of visualizations for parallel/distributed systems has not been realized. Constructing meaningful and useful visualizations remains an art, and system complexity seems to increase faster than tools can be developed and visualizations created. While some tools (e.g., [51][74][71][126][119]) are flexible enough to provide meaningful information across multiple systems, the construction of custom visualizations remains difficult and expensive. To date, most visualization systems have been designed based on a narrow set of requirements or for a specific

context, little re-use of existing visual displays has been possible, and few well-understood forms, abstractions, and concepts have been applied to constructing visualizations.

Tables 1 summarizes a representative subset of performance metrics, tools, and displays in use today for cluster/grid.

Table 1. Cluster and Grid Performance Visualization (Representative Subset)

<i>Displays</i>	<i>Possible Metrics</i>	<i>Usage Examples</i>	<i>Selected Tools</i>
Bar chart	CPU load, TCP bandwidth & latency, memory free & used, disk size & used	Displaying a number of retransmissions over time	Pablo [83], NetLogger Visualization [76]
Line chart	CPU load, TCP bandwidth & latency, memory free & used, disk size & used	Displaying CPU load over time	NetLogger Visualization [76], Vampir [117], TAU [71], ParaGraph[51], Pablo [83]
Pie chart	CPU system time, user time	Displaying memory free, used, cached, and swapped	Vampir [117]
Scatter plot	Memory size, execution time	Displaying message size versus message processing time	Pablo
Kiviat diagram	CPU load, process load	Utilization of N processors	TAU
Histogram	Message, memory, I/O metrics	Displaying a number of messages, a number of cache misses, and a number of I/O accesses	Jumpshot [126]
Gantt chart	Process status	Displaying states of N processes, busy, idle, overhead, over time	TAU, Vampir, Pablo, Jumpshot, ParaGraph
Interaction matrices	Network bandwidth & latency	For N processors, in NxN matrix, interaction between the pair represented by color, shape, size or intensity of each cell	Vampir, Pablo, ParaGraph
Network graph	Node status, network bandwidth	Displaying running status of N nodes and their links to each other	Pablo, ParaGraph, Vampir

<i>Displays</i>	<i>Possible Metrics</i>	<i>Usage Examples</i>	<i>Selected Tools</i>
Execution graph	Process status, call status	Displaying communication events among N processes over time	Vampir, Jumpshot, NetLogger Visualization
Source code	Source code metrics	Highlighting loop nests and procedure calls in a source code	Pablo, SeeSoft [37], SCALEA [113]

These tables highlight the similarities between tools and systems in the types of displays and their usage. They, however, also underscore the needs identified above with different tools being used for different resources and analyses, making it nearly impossible to support wide-scale system visualization.

1.2 Motivation

Complex parallel and distributed systems must be tuned to achieve good performance. By monitoring the behavior of resources, a system designer may identify bottlenecks and non-optimal design solutions and then modify some aspect of the system's design to improve performance. Visualization has been widely accepted as a means to deal with large-scale data sets including dynamic and multivariate performance data. Many visualization techniques have proven effective for understanding complex parallel systems. The benefit of visualizations, however, is not limited to complex parallel systems. Since grid [41] and web technologies are enabling distributed systems to be interconnected, the complexity of their resource hierarchy and heterogeneity is being increased substantially.

Most visualization techniques, however, have focused on specific types of performance events at a certain level of the system. These techniques have been developed in isolation to solve specific problems, with little consideration for the

interactions and dependencies within the larger system. In contrast, a distributed application in heterogeneous environments (such as grids) often is engaged with various types of performance problems during its execution across multiple levels and platforms, and often needs to be dynamically configured to accommodate newly available resources.

To illustrate the situation, consider a distributed collaborative virtual reality (VR) application that concurrently executes on a SGI Onyx at ISU and a Linux cluster at NASA. Both systems are connected by a high-speed vBNS network. Running the application within a collaborative environment, scientists at both sites are sharing the same virtual object of a space station; the object is being projected on the C6 CAVE at ISU and on an Immersadesk at NASA. During this collaboration session, scientists are trying to decide the best position of a solar panel, which is attached to the space station, for its maximum exposure to the sun. Let us consider the following performance scenario. As the ISU scientist moves the solar panel object to the top of the station cockpit object, the scientist at NASA experiences an unexpected latency during rendering the movement of the solar panel object. The latency might be caused by a particular rendering routine. In this case, we can pinpoint the problem by monitoring a specific rendering process. However, suppose the latency is associated with several causes; e.g., an unnecessary retransmission caused by an inappropriate TCP buffer size results in a delay in rendering. Monitoring a particular resource would not be enough. Instead, multiple sub-analyses across resources and platforms should be conducted and integrated.

Monitoring such a distributed application is not trivial. It requires an extensible monitoring architecture for accessing distributed performance data and system-level visualizations for observing data correlation of multiple resources [41]. Several monitoring frameworks have been developed to meet these requirements [87][106][124][125]. Their visualizations, however, still rely on platform- and resource-

specific tools that are not sharable. Performance visualization should adapt for the complexity and heterogeneity of system, such that what is viewed and how it is viewed are not constants. The visualization should be consistent, reusable, and composable to support performance analysis associated with various resources across different platforms.

This research is motivated by several limitations in current performance visualization techniques: (1) they are limited to specific resources; (2) they are not sharable; and (3) they do not support dynamic system-level visualizations.

1.3 Problem Statement

Performance visualization tools typically focus on specific resources or levels in the system. In contrast, distributed applications are executing in grid environments having greater heterogeneity and transience. New performance visualization technology is needed that scales with system complexity, where what is viewed and how it is viewed are not constants. To address this, we have developed a performance monitoring framework called Uniform Resource Visualization (URV) [66]. URV addresses the following needs in performance visualization for heterogeneous and distributed environments:

Need for Standard Visualization Services: A complex computing system consists of heterogeneous, coordinated resources, where resources are any logical or physical elements that comprise the system. In a grid environment, a resource may be, for instance, a network router or an application subroutine. Visualizing these resources and their interactions from a system-level perspective requires a visualization and instrumentation

that is uniform across different resources. The framework supports the integration of information about different resources. Resource developers/vendors should be able to provide custom visualizations of a resource, and these visualizations should be designed consistently for interactions with other software components. Most performance visualizations, however, have been developed in isolation, with little consideration for interactions and dependencies within a larger framework. In particular, separately developed visualizations often end up having completely different interfaces even though they support the same interactions. Several visualization toolkits [2][56] provide uniform interfaces to define interactions between sub-modules of a visualization. However, uniform interfaces for interaction between a visualization and other components have not been proposed. This research addresses the issue of standards to prescribe interfaces for accessing and managing visualization components

Need for Sharing Visualization knowledge: It is not unusual that performance visualizations from different domains have a lot in common (e.g., single metric-based views or generalized views that can be customized to different system levels). Unfortunately, their crucial design features are more likely to have been reinvented than reused from one another. Consequently, there is a need to reuse both knowledge about visualization and specific visualizations (i.e., software components). A visualization developer/user should be able to search for a visualization that meets the needs of an analysis problem. In the information visualization area, visualization taxonomies have been developed [70][98][91][105]. Such taxonomies help visualization developers organize the design features according to the data domain of interest. It is difficult to document a software component in unambiguous, and classifiable terms; a description of a component must be translated into a concrete specification. A specification and

indexing scheme must be applied to classify the features and guide the search for components. This research addresses the issue of reuse with an ontology-based approach that will provide the capability to search for visualization components and find matches according to the classification.

Need for Visualization Composition: A system-level visualization should present data correlation between multiple resources. The simplest way to construct a system-level visualization is to aggregate individual visualizations from each resource, placing them side-by-side in a window. This simple method suffers two problems. First, the separate visualizations may not be synchronized, which may cause users to draw incorrect correlative inferences about the performance of the system. Second, any performance information that is a function of the performance of multiple resources in cooperation will not be directly visible. The first problem concerns the need for spatial/temporal synchronization among separate visualizations. The second problem concerns the need for abstraction and layering in a visualization. These two problems are fundamental obstacles to the design of a system-level visualization. Several performance visualization tools [103][76] support system-level visualization based on temporal synchronization. For instance, as shown in Figure 1, Netlogger visualization [76] allows visualizing multiple performance data in a single view. Their system-level views, however, are limited to simple aggregation of predefined views (e.g., line chart). Such system-level views are not effective to provide meaningful observation if a number of aggregated views grows big; users suffers from too many individual views, which could require additional analysis on the views themselves. A system-level view needs to be in a concise form.

To monitor complex computing systems, dynamic creation of integration of performance information is necessary. This research addresses information integration issues associated with system-level views.

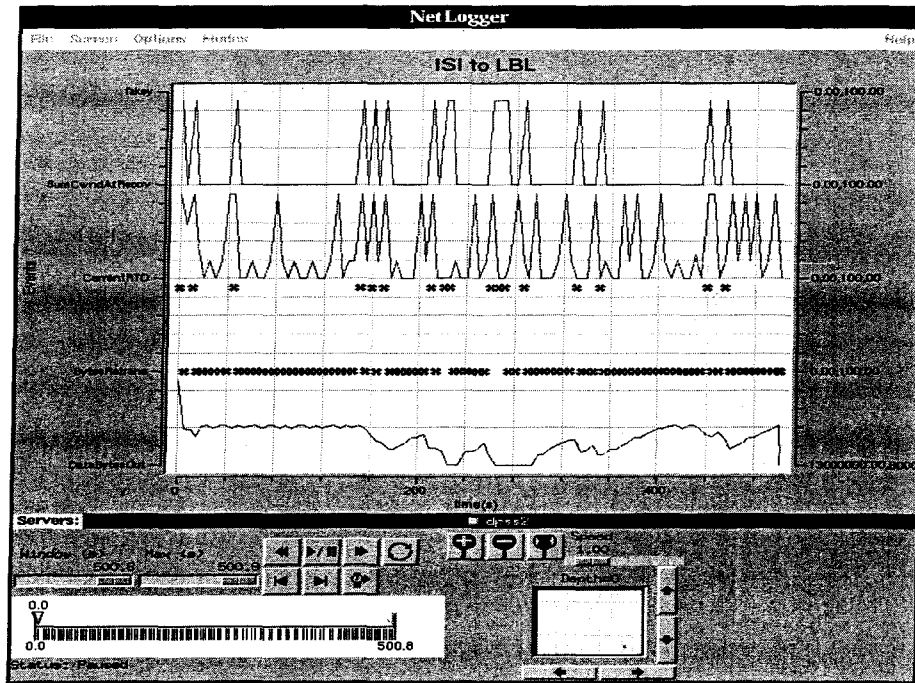


Figure 1. Netlogger Visualization [76] supports system-level visualization which is only based on temporal-based synchronization. It allows add more data, but the view type is fixed.

Need for New Visualization Technology: The problems of uniformity and reusability can be partially addressed with an appropriate software engineering approach to the visualization system. That solution, however, will fall short without a structured approach to visualization creation and the ability to share visualization knowledge. Capturing visualization knowledge is a hard problem with limited coverage thus far in the research community. The problem of composition is not merely one of software engineering, yet composition is the key to system-level visualization. Visualization composition is a real challenge, in part because it requires integration at several levels of

information, including visualization design. The research community has only begun to leverage new software technologies for advanced visualization systems. The concepts

Thesis Statement Performance visualization in heterogeneous and distributed systems should be supported by standard visualization service, sharable visualization knowledge, and system-level views. Component technology can be applied in a novel manner for creating new performance visualizations. In particular, uniform interfaces are a means to access and manipulate visualizations consistently. In addition, a representation of visualization knowledge supports reuse of visualizations. Finally, information integration into new system-level views is based on composition of visualization components.

behind visualization design are mature enough to build them into new visualization systems, so that users can focus on “what” they want to see, not “how” to see it.

The central research issues addressed in this research emphasize the need for uniformity, reusability, and composition in performance visualization systems. This research demonstrates new performance visualization technology applied to a complex computing system.

1.4 Objectives

The objectives of this dissertation research are summarized as follows:

- **To reuse visualization components:** One objective is to define visualization components so that they are reusable. The key is to provide a methodology for systematic creation of visualization components. In part, interfaces for their access and manipulation need to be uniformly defined, including methods, parameters, and an interaction protocol. The interface definitions need to support a wide range of performance visualization situations.
- **To represent performance visualization knowledge:** This research should describe performance visualization knowledge so that it is sharable. To do this, current visualization representations associated with data-model based taxonomies should be extended to accommodate visualization knowledge.
- **To create system-level visualizations dynamically:** The ultimate objective of this research is to provide a systematic methodology for integrating information into system-level visualizations. To do this, visualization components need to be compatible both syntactically and semantically. Visualization components can be used in multiple visualizations without losing its syntactic and semantic meaning. Likewise, a new visualization that inherits knowledge of multiple visualizations can be created. This research should provide the principles and methods for integrating performance views.

1.5 Contributions

The contributions of this research take three forms: new or enhanced qualities of performance visualization, new methodology, and new software.

A. Qualities of a performance visualization

1. *This research extends current visualization representation techniques to the field of performance visualization.* This work provides a way of representing performance visualization based on its knowledge, which improves reusability of visualizations.

B. Visualization methodology

2. *This work creates a methodology for designing reusable visualization components.* This methodology addresses the heterogeneity in creation of visualization components. A visualization component which is created based on the methodology is able to be consistently accessed and integrated into the visualization system.

3. *This work provides a methodology for creating system-level views by composition.* The principles and methods of composition are provided. It also includes a new spreadsheet-based approach to provide a structured way of applying composition operation to multiple performance views

C. Visualization software

1. *This research presents a spreadsheet-based performance visualization tool.* The spreadsheet-based performance visualization tool provides a structured way of observing, exploring, understanding, and managing performance problems.

2. *This research presents a performance visualization framework.* The framework consists of the implementations of visualization methodologies, which provide users with software modules to help create a performance visualization system in a consistent way.

1.6 Organization of Dissertation

The following dissertation is organized as follows. Chapter 2 describes a visualization model of URV, which presents basic elements of a performance visualization. In Chapter 3, the representation of visualization knowledge is described. This knowledge representation supports systematic searching of visualization components for specific goals. We address this issue in Chapter 4 by providing a visualization component creation methodology. Visualization component, which is one of the elements, is the main focus of this research. How to develop a consistent and reusable visualization component is an very important issue for system-level visualization. The methodology consists of several systematic steps which lead to reusable visualization component development. Chapter 5 defines the composition methodology for creating a system-level view. We present different types of visualization composition and based on them, we provide different strategies of composition. Chapter 6 presents a new spreadsheet-based visualization tool that we designed for facilitating creation and composition of visualization components. Chapter 7 presents a performance visualization system, which consists of services implementing the visualization methodologies, and a case study that demonstrates our methodologies. Chapter 8 provides some related works that have influenced this research. Chapter 9 then concludes the dissertation by summarizing research contributions and future work.

CHAPTER 2. VISUALIZATION MODEL

In this Chapter, we describe our visualization model. The model defines the elements of a performance visualization, and their behaviors and interactions. The creation of performance visualization is a process of defining the scope and behavior of each element, and interactions among them.

2.1 View

A view is a visual representation of information. In particular, a performance view shows performance information such as hardware performance and computation behaviors. It is used to evaluate performance, verify correctness, diagnose problems, and gain insight into structure and execution behavior. The performance view consists of the following elements:

- **Data (D):** A data specifies information to be visualized. Three most commonly used data types are quantitative, ordinal, and nominal [70]. Nominal data types are assemblages of symbolic names, typically unordered. For example, the names of the explorers, such as Gallileo, Columbus, Magellan, etc., form a nominal data set. Ordinal data types are rank ordered only. The ordering of the data does not reflect the magnitudes of the differences. A typical example of an ordinal date set is the names of the calendar month, January through December. Quantitative data present real numbers

- **Graphical Primitive (GP):** A graphical primitive is a computer generated visual entity, such as a line, dot, bar, and circle. It specifies how to map data to graphical primitive [70].
- **Visual Design (VD):** Visual design presents the structure and relationships within a data set in an effective format [70]. It describes how to structure and layout visual elements. Bar chart design and matrix view design are examples.
- **View (V):** A view consists of multiple visual elements with a visual design.
- **Visual Element (VE):** A visual element is a graphical primitive or view that presents data.
-

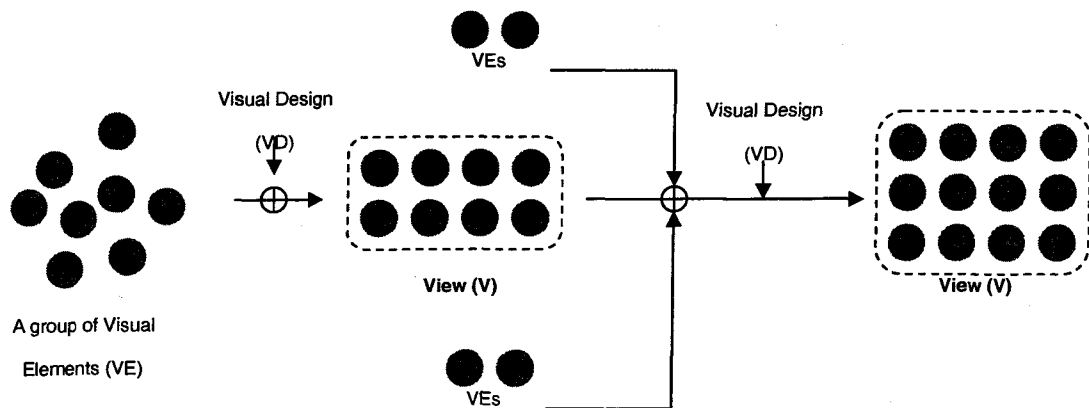


Figure 2. View and Visual Elements

Using the elements, we can establish the following relationship between elements:

- $VE = VE \text{ VE} \mid GP + D \mid V \mid \varepsilon$
- $V = VE + VD \mid \varepsilon$
- $VD = \text{matrix view design} \mid \text{bar chart design} \mid \dots$
- $GP = \text{Line} \mid \text{Bar} \mid \text{Dot} \mid \text{Circle} \dots$
- $D = \text{quantitative data} \mid \text{ordinal data} \mid \text{nomial data}$

The view itself can be a visual element that can be one of elements of another view. Figure 2 depicts a view and its relationship with visual elements and visual designs.

2.2 Uniform Resource Visualization (URV)

Uniform Resource Visualization (URV) is a new visualization framework, focusing on integration of performance information into system-level visualizations. As depicted in Figure 3(a), typical performance visualization environments are based on a one-to-one relationship between instrumentation and visualization. That is, there is often a single source or format of performance data prescribed for a particular visualization. Although there are exceptions, e.g., the model of Pablo [83], a user typically sees a graphical display that came packaged with the instrumentation. This is the case even with a many-to-one relationship, in which a basic graphical display type is paired with all performance data (as in Paradyn [74]). The availability of new performance data often means that new visualizations will need to be programmed specifically for that tool environment. There is little chance that a visualization can be used with another tool, even though many of its attributes would satisfy the end-user needs.

The main goal of URV is to extend the usability of visualization by providing a many-to-many relationship between instrumentation and visualization. The many-to-many relationship supports the special cases of one-to-many and many-to-one relationships. In case of a one-to-many relationship, a single source of performance data can be viewed in several ways. This lets a user choose a perceptually effective visualization. On the other hand, a many-to-one matching enables a single visualization to be reused across several sources of performance data. For instance, a line chart can display CPU load over time,

round-trip message time over time, etc. In addition, since performance views in URV are created in a uniform and consistent way, the views have syntactic and semantic compatibility with each other. A graphical representation in one performance view can be reused in a second view without losing its syntactic representation and semantic meaning. This feature enables the integration of performance views, whereby a new visualization is created by composing multiple visualizations.

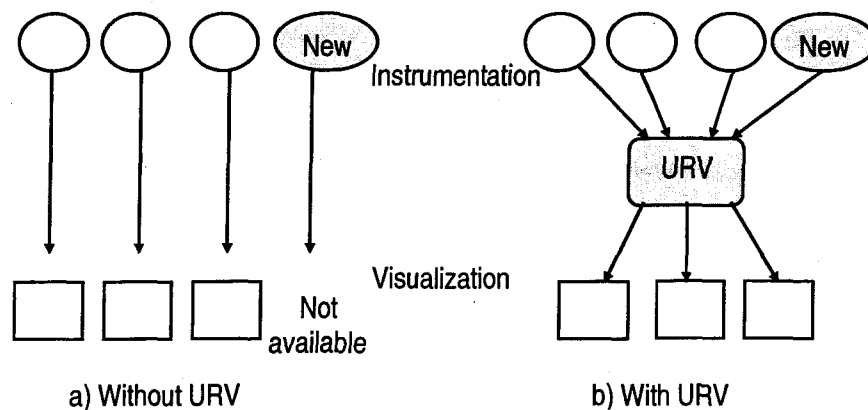


Figure 3. Performance visualization with/without URV

2.3 Performance Visualization in URV

A performance visualization in this research consists of a resource part and a visualization part [64], as illustrated in Figure 4. The resource part of the visualization consists of a *Resource* and a *Resource-Monitoring Component (RMC)*. A resource is a source of performance monitoring. It may be a physical entity (e.g., CPU, Network link, log file) or a logical entity (e.g., process).

An RMC provides instrumentation, data collection, and interaction with consumers of performance data. It separates visualization from resource. There are two types of

performance data collected by RMC: *metric data* and *event data*. The metric data presents the status, throughput, and usage of resources. On the other hand, the event data presents a series of changes/happenings during resource running/execution (e.g., send & receive events during data transmission). Both types of data consist of timestamp, the name of metric/event, and quantity value (e.g., Utilization % value). The quantity value can be optional for event data.

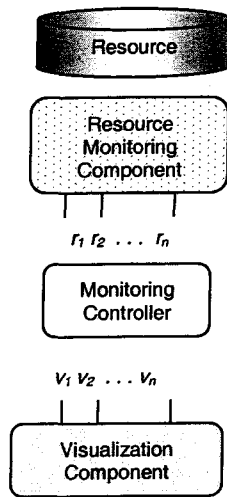


Figure 4. URV View

The semantics of an RMC (i.e., what RMC measures or/and monitors) can be represented with a Performance data type (**P**), a Resource type (**R**), and a target Node (**N**):

RMC is one of $\{ \langle n, r, p \rangle \mid n \text{ is a set of node IPs, } r \text{ is a subset of } R, p \text{ is a subset of } P, \text{ where } R \text{ is a total set of resource types, } P \text{ is a total set of performance data types} \}$

A single RMC $\rightarrow \langle n, r, p \rangle$

For instance, suppose there is a RMC_i that monitors a CPU load of the node (123.121.133.98). Then, this RMC_i can be represented as:

$$\text{RMC}_i \rightarrow \langle \{123.121.133.98\}, \{\text{CPU}\}, \{\text{Load}\} \rangle$$

In URV, we assume that all RMCs are unique; there are no two RMCs that collect the same performance data of the same resource at the same node. In addition, we have restricted resource and performance data types to the most commonly used ones. The detailed list is described in Chapter 3.

The visualization part is termed a *visualization component (VC)*. A VC is a software component implementation of a *View (V)*. In addition, the VC also contains information about Analysis type (A). An analysis type specifies the purpose/usage of visualization. For instance, one visualization is very useful for data trending analysis while the other visualization for observing interactions.

The semantics of VC can be represented with A and VD:

VC is one of $\{ \langle a, vd \rangle \mid a \text{ is a subset of } A, vd \text{ is a subset of } VD, \text{ where } A \text{ is a total set of analysis types, } VD \text{ is a total set of visual designs} \}$

$$VC \rightarrow \langle a, vd \rangle$$

For instance, a VC_i , which implements a barchart view and supports observing the status of resource, can be represented as:

$$VC_i \rightarrow \langle \{ \text{status} \}, \{ \text{BarChartView} \} \rangle$$

Likewise, a VC_j , which implements a network view and supports status & interaction monitoring:

$$VC_j \rightarrow \langle \{ \text{status}, \text{interaction} \}, \{ \text{NetworkView} \} \rangle$$

Based on the definitions of RMC and VC, a URV view can be further defined by coupling RMC with VC:

$$\begin{aligned} \text{Performance Visualization} &\leftrightarrow \text{RMC} + \text{VC} \\ &\leftrightarrow \langle n, r, p \rangle + \langle a, vd \rangle \\ &\leftrightarrow \langle n, r, p, a, vd \rangle \end{aligned}$$

For instance, a performance visualization, which visualizes the status of CPU load at the node, 123.121.133.98 using a Barchart view, is represented as:

$$\text{URV View} \rightarrow \{ \langle \{123.121.133.98\}, \{CPU\}, \{Load\}, \{Status\}, \{BarChartView\} \rangle \}$$

A *monitoring controller* defines component interaction via a set of services that are provided by one component and required by another component. Each component (RMC, VC) specifies services comprised in its interface to other components, e.g., $r_1 r_2 \dots r_n$ and $v_1 v_2 \dots v_n$ in Figure 4. Accessing components via the interfaces, the monitoring controller controls performance visualization. It could be a user customized program which controls monitoring. For instance, a GUI console that hosts instrumentation and performance visualization is a good example.

In order to create RMC, we can convert existing tools or software modules. For instance, network monitoring sensors in NWS [124] and monitoring tools in NetLogger [108] can be wrapped to create RMCs. In the meantime, this research does not address methodology of RMC creation. Instead, the research focuses on VC.

URV views can be implemented by mapping onto existing software architectures, e.g., the Grid Monitoring Architecture (GMA) [109] depicted in Figure 5(a). As depicted in Figure 5(b), an RMC and a VC in URV correspond to a producer and a consumer in GMA, respectively, in the sense that the VC consumes performance information that the RMC produces. Besides, a GMA-based monitoring tool could apply selected URV components, such as a specific VC for visualization.

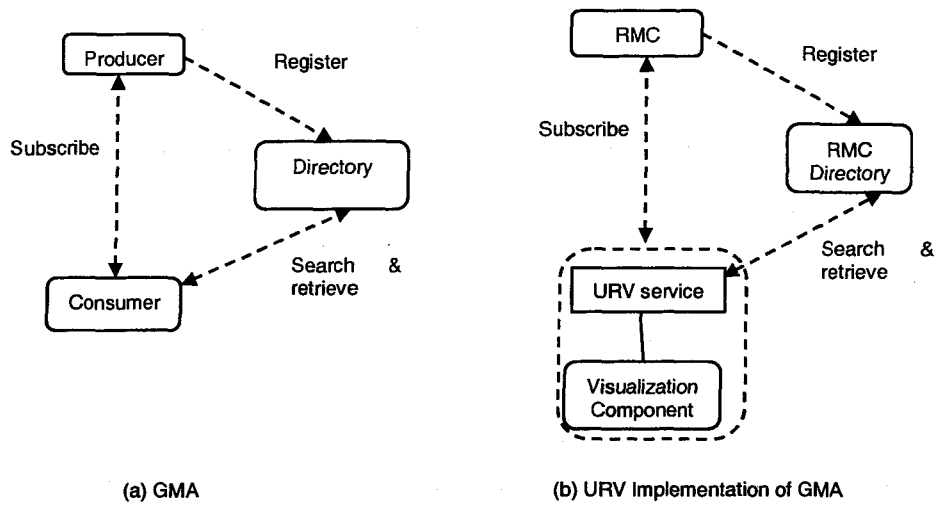


Figure 5. GMA and URV Implementation

2.4 Summary

This Chapter provided basic information about performance visualization. We first described the different aspects of a performance view, including a data, graphical primitive, visual design, and visual element. Based on the basic information, we have further defined a performance visualization model, which defines the elements and their interactions of performance visualization. The elements include resource, resource monitoring component, visualization component, and monitoring controller. In this Chapter, we formalized the representation of each element, which is used in the rest of thesis. The proposed visualization model can be easily mapped to existing consumer/provider based monitoring architecture.

CHAPTER 3. PERFORMANCE VISUALIZATION KNOWLEDGE

This Chapter describes a representation of performance visualization knowledge. The knowledge representation is machine-interpretable, which allows software agents to conduct a query-answer process. The visualization knowledge is a key deliverable to improve reusability of visualizations.

3.1 Performance Visualization Knowledge (PVK)

Two approaches are commonly used toward reusable visualization in the area of information visualization: visualization taxonomy and automatic visualization. Researchers have constructed taxonomies of visualization techniques by examining the data domains that are compatible with the techniques [21][101][114]. The taxonomies help developers quickly identify various techniques that can be applied to the domains of interest. The taxonomy approach provides a basis to identify relevant visual design content, however, applying and implementing a technique is left to the developer. Automatic visualization methods [17][26] help developers create a view by suggesting an intermediate view based on predefined design principles at each stage of the visualization process. Tuning of the intermediate view eventually leads to a final view. This technique narrows the design space and has proven useful for non-visualization experts.

This research attempts to reuse visualization based on its knowledge. Knowledge denotes the fact of knowing something with familiarity gained through experience or learning. Knowledge of visualization describes which goals work best with a visualization [47]. In particular, knowledge of a performance visualization knowledge, termed

Performance Visualization Knowledge (PVK), represents 'what types of performance monitoring situations have been well matched with a certain visual design'. In URV, PVK presents the properties of performance information that are well matched with a specific visual design. In particular, PVK presents the following information:

- What resources (R) can be best visualized by VC?
- What types of performance event (or/and) metric data (P) of the resource(s) can be best visualized by VC?
- What types of analysis (A) does VC support?

For convenience, we can represent a single PVK as a combination of those information: (A, R, P), where A is a set of analysis types, R is a set of resource types, and P is a set of performance data types.

Reuse of visual design implies that, using the same design, we are guaranteed to create multiple views that have the same syntactic graphical presentation and perceptual understanding. Furthermore, reusing the same visualization knowledge guarantee the same semantic content.

Describing PVK is one of main goals to support reusability of visualization. The description of visual design and visualization knowledge must meet the following requirements:

- Versatility: A description should represent a group of multiple views based on syntactically compatible visual designs. Creating multiple instances from a single description reduces the number of descriptions that must be maintained.
- Extensibility: A description should allow the addition of new content.

- Hierarchy: A description must support a hierarchical representation.

3.2 Performance Visualization Ontology (PVO)

In order to meet the requirements defined previously, we have exploited an ontology-based approach. Ontology defines a common terminology for users who need to share certain information in a domain. It contains the definitions of basic concepts in the domain and relations among them [78]. Ontology provides a consistent way of sharing domain specific knowledge, which leads to reuse of well-recognized solutions.

The Performance Visualization Ontology (PVO) is a collection of Performance Visualization Knowledge (PVK)s. It describes which visual design works best with which monitoring goals, which combinations of visual design provide the most effective performance view.

There is no single correct way to develop ontology for a domain. Depending on a specific usage of ontology, it is possible to have multiple ontologies, which model the same domain. In general, ontology development is necessarily an iterative process [78]. In this research, we model the performance visualization domain, focusing on the characteristics of performance monitoring scenarios and the corresponding visual designs.

There could be virtually unlimited performance monitoring problems and visual designs to be modeled. It is not desirable to model them individually, which leads to unnecessary complexity in ontology. Instead, we have restricted the scope of domain to the most commonly used monitoring situations and visual designs in high-performance distributed computing environments. For instance, the monitoring related terminologies in PVO were borrowed from several on-going standards in the performance monitoring community, such as [84] and [109]. Since one terminology might be used in a different

way in the other areas, we assume that users of PVO are familiar with the terms and their meanings in the area of high-performance distributed computing.

3.3 PVO Representation

To represent the PVO, we use a Resource Description Framework (RDF)-based ontology representation, such as Darpa Agent Markup Language (DAML) [31]. DAML provides plentiful constructs to create ontology and to markup information so that it is machine readable and understandable. In DAML, we represent a basic knowledge model with a combination of classes (conceptualization), a class hierarchy, properties of class, and instances. Thus, building a knowledge model with DAML is essentially a process of defining classes, building a class hierarchy, defining properties, and creating instances. In this research, we focus on the systematic mechanism of representing and utilizing visualization knowledge rather than the completeness of knowledge modeling.

3.3.1 Class and Class Hierarchy

A class is a main concept in a domain. A class can have sub classes that represent more specific concepts than a class (See Figure 6). For instance, a class of performance data represents all performance data. A class of performance metric data is a sub class of performance data, which represents all number-based metric data. Further, a load could be an instance of performance metric class. All sub classes and instances inherit the properties of super class. All classes, sub classes, and their hierarchies in PVO are as follows:

- ANALYSIS_TYPE class: represents the types of analysis that can be conducted by performance visualizations. For instance, one visualization is very useful to observe a trend while another visualization is useful to observe interaction between resources.
- RESOURCE class: represents target resources to be monitored.
- PERFORMANCE_DATA class: represents performance data associated with a target resource.
 - PERFORMANCE_EVENT_DATA class: represents performance event data. E.g., send event, receive event
 - PERFORMANCE_METRIC_DATA class: represents performance metric data. E.g., load, utilization
- VISUAL_DESIGN class: represents selected visual designs
 - BAR_CHART_DESIGN class: represents bar chart designs that show quantitative values
 - LINE_CHART_DESIGN class: represents line chart designs which show quantitative values
 - PIE_CHART_DESIGN class: represents pie chart designs which show the contribution of each performance data with different pie sizes
 - SCATTER_PLOT_DESIGN class: represents scatter plot designs which show the correlation between pairs of performance data
 - GANTT_CHART_DESIGN class: represents gantt chart designs which show the progress of event(s) over time
 - SPACE_TIME_DIAGRAM_DESIGN class: represents space time diagram designs that shows the progress of event(s) and interaction between events over time

- **MATRIX_VIEW_DESIGN** class: represents interaction matrix chart designs that show interaction between the associated pairs
- **NETWORK_VIEW_DESIGN** class: represents network graph designs that show structure and relationship between objects

```

<!-- Class Definition -->
<daml:Class rdf:about=#PERFORMANCE_DATA>
  <rdfs:label>PERFORMANCE_DATA</rdfs:label>
</daml:Class>

<daml:Class rdf:about=#PERFORMANCE_METRIC_DATA>
  <rdfs:label>PERFORMANCE_METRIC_DATA</rdfs:label>
  <!-- Sub class definition -->
  <rdfs:subClassOf>
    <daml:Class rdf:about=#PERFORMANCE_DATA/>
  </rdfs:subClassOf>
</daml:Class>

<!-- Example: Bar chart design class -->
<daml:Class rdf:about="#BAR_CHART_DESIGN">
  <rdfs:label>BAR_CHART_DESIGN</rdfs:label>
  <rdfs:subClassOf>
    <!-- Bar chart design is a sub class of VISUAL_DESIGN
class -->
    <daml:Class rdf:about="#VISUAL_DESIGN"/>
  </rdfs:subClassOf>

```

Figure 6. DAML representation of a class (Performance data) and sub class (Performance Metric Data), and the example of BarChart visual design class

3.3.2 Properties

Properties in ontology provide more information about classes. In particular, in PVO, we focus on detailing a visual design class for matching a visual design to right

monitoring situation(s). For instance, which analysis types are supported by a particular visual design? Which resources can be visualized with the visual design? , etc. We have defined the following three properties:

- *supportAnalysisType*: represents which analysis types are supported by the visual design
- *supportPerformanceData*: represents which performance data types are supported by the visual design
- *supportResource*: represents which resource types are supported by the visual design

Figure 7 shows the properties of the visual design class and relationships with other classes. In addition, Figure 8 shows the DAML representation of property definition.

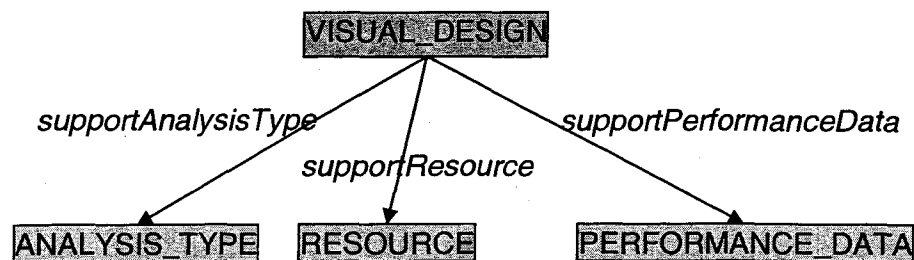


Figure 7. Visual Design class and Its properties

```

<!-- Definition -->
<daml:ObjectProperty rdf:about=#supportAnalysisType>
  <rdfs:label>supportAnalysisType</rdfs:label>
  <rdfs:range>
    <!-- A target range of property -->
    <daml:Class rdf:about=#ANALYSIS_TYPE/>
  </rdfs:range>
</daml:ObjectProperty>

```

Figure 8. DAML representation of supportAnalysisType Property

3.3.3 Instances

This section describes instances of each class. An instance presents a specific case of a class. It establishes a “is-a” relation with a class; an instance is a type of class. In addition, instances inherit the properties of the parent class. Our list of instances does not necessarily present every possible instance of each class. Instead, it focuses on the most commonly used ones in the area of performance monitoring.

Table 2 summarizes the instances of ANALYSIS_TYPE, RESOURCE, and PERFORMANCE_DATA. Based on these instances, we then present the instances of visual design, which correspond to several well-recognized performance visualization usages in high-performance and distributed computing.

Table 2. Instances of Classes of Analysis, Resources, and Performance Data

<i>Class</i>	<i>Sub class</i>	<i>Instance</i>	<i>Note</i>
ANALYSIS_TYPE		<i>Trend</i>	Observes data trending
		<i>Status</i>	Observes status of resources. (e.g., process status: running, idle)

<i>Class</i>	<i>Sub class</i>	<i>Instance</i>	<i>Note</i>
		<i>Relationship</i>	Observes relation between performance data (e.g., correlation)
		<i>Comparison</i>	Compares performance data
		<i>Structure</i>	Observes the structure of resources. (e.g., network hierarchy)
		<i>Interaction</i>	Observes interaction between pairs of resources (e.g., communication between processes)
		<i>Contribution</i>	Observes contribution of each performance data
RESOURCE		<i>Process</i>	Process
		<i>Processor</i>	CPU
		<i>Memory</i>	Storage
		<i>Disk</i>	Storage
		<i>Link</i>	Network link
		<i>Software</i>	Application
PERFORMANCE_DATA	PERFORMANCE_EVENT_DATA	<i>Send</i>	Send event (e.g., data sent)
		<i>Receive</i>	Receive event
		<i>Start</i>	Start processing/running/executing
		<i>Stop</i>	Stop
		<i>Ack</i>	Acknowledgement
		<i>Read</i>	Read data block(s)
		<i>Write</i>	Write data blocks(s)
		<i>Idle</i>	Idle
		<i>Busy</i>	Busy event (e.g., a process is computing)
	PERFORMANCE_METRIC_DATA	<i>Available</i>	e.g., available memory
		<i>Utilization</i>	e.g., processor utilization
		<i>Bandwidth</i>	e.g., network bandwidth
		<i>Load</i>	e.g., processor load
		<i>Latency</i>	e.g., network latency (round trip time)

<i>Class</i>	<i>Sub class</i>	<i>Instance</i>	<i>Note</i>
		<i>PacketLoss</i>	e.g., network packet loss
		<i>NumberOfHops</i>	e.g., a number of hops between two hosts
		<i>Size</i>	e.g., disk size
		<i>Used</i>	e.g., memory used

Note that resources such as **processor** and **memory** are instances of the **RESOURCE** class instead of sub classes. It implies that PVO considers their all variations as the same instance. For instance, a visualization for AMD64 processor utilization is not necessarily different from the one for Intel processor utilization.

Each instance of **VISUAL_DESIGN** class presents a specific usage of visual design in performance monitoring. Table 3 shows the selected instances of visual design, each of which specializes its parent class with target instances (in Table 2) of the properties.

Table 3. Selected Instances of Visual Design Class

<i>Class</i>	<i>Sub Class</i>	<i>Instance</i>	<i>Property</i>	<i>Visualization</i>
VISUAL_ DESIGN	BAR_CHART_DESIGN	<i>BarChart_1</i>	<i>supportResource: Link</i>	A bar chart that observes link bandwidths over time
			<i>supportPerformanceData: Bandwidth</i>	
			<i>supportAnalysisType: Trend</i>	
	LINE_CHART_DESIGN	<i>LineChart_1</i>	<i>supportResource: Processor</i>	A line chart that observes processor loads over time
			<i>supportPerformanceData: Load</i>	
			<i>supportAnalysisType: Trend</i>	
	PIE_CHART_DESIGN	<i>PieChart_1</i>	<i>supportResource: Memory</i>	A pie chart that shows how much memory is available vs. not available
			<i>supportPerformanceData: Available</i>	
			<i>supportAnalysisType: Contribution</i>	
	SCATTER_PLOT_DESIGN	<i>ScatterPlot_1</i>	<i>supportResource: Processor, Memory</i>	A scatter plot that shows correlation between processor load and memory usage
			<i>supportPerformanceData: Load, Available</i>	
			<i>supportAnalysisType: Relationship</i>	
	GANTT_CHART_DESIGN	<i>GanttChart_1</i>	<i>supportResource: Process</i>	A gantt chart that shows process status (busy, idle) over time
			<i>supportPerformanceData: Busy, Idle</i>	
<i>supportAnalysisType: Status</i>				
SPACE_TIME_DIAGRAM_DESIGN	<i>SpaceTimeDiagram_2</i>	<i>supportResource: Process</i>	A space time diagram that shows status of multiple processes and communication between them	
		<i>supportPerformanceData: Send, Receive, Busy, Idle</i>		
		<i>supportAnalysisType: Status, Interaction</i>		
MATRIX_VIEW_DESIGN	<i>MatrixView1</i>	<i>supportResource: Link</i>	A matrix view that shows bandwidths between pairs of links	
		<i>supportPerformanceData: Bandwidth</i>		
		<i>supportAnalysisType: Interaction</i>		
NETWORK_VIEW_DESIGN	<i>NetworkView_1</i>	<i>supportResource: Link, Processor</i>	A network graph that shows distributed nodes. Each node in a view presents the existence of processor. Colors of edges in a view present link bandwidth	
		<i>supportPerformanceData: Bandwidth</i>		
		<i>supportAnalysisType: Structure, Status</i>		

3.4 Ontology Query

PVO, once represented in DAML, can be queried using the DAML Query Language (DQL) [30]. DQL is a formal language and protocol for software agents to perform a query-answering process with knowledge represented in DAML. A DQL query necessarily includes:

- A query pattern that is a collection of DAML sentences in which some of the literals or/and Uniform Resource Identifiers references (URIrefs) have been replaced by variables.
- An answer Knowledge Base (KB) pattern that is a KB reference, a list of KB references, or a variable
- A must-bind variables list and a may-bind variables list. Answers are required to providing bindings for all must-bind variables, may provide bindings for may-bind variables

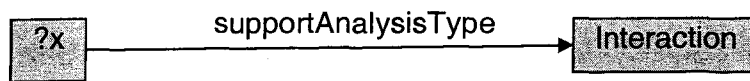


Figure 9. RDF Triple model for What Supports Interaction Analysis Type

In essence, a query-answering of ontology is a process of reasoning and matching given variables to the right class(es) or/and instance(s) in the ontology. To represent a query, for example, “What visual designs do support an interaction analysis type? “, first

we create an intermediate form of the query in a RDF triple model. Figure 9 shows the RDF model of the query. $?x$ presents a must-bind variable and tkb , target knowledge base. In addition, its DAML representation is shown in Figure 10. Figure 11 shows the sample query result of “what visual designs does support interaction analysis type?”.

3.5 Summary

This Chapter provided the definition and principles of performance visualization knowledge. The performance visualization knowledge presents ‘what types of resources, performance data type, and/or analysis type have been well presented with a certain visual design’. We presented an ontology-based knowledge representation, which results in Performance Visualization Ontology (PVO). PVO defines the concepts (classes) and properties of performance visualization that are basis in categorizing performance visualization based on the knowledge. The PVO is a key item that supports reusability and composition of visualization. With the PVO, users can obtain well-recognized visual designs for observing a particular performance problem. Users do not need to reinvent a new visual design unless a target problem domain is completely unrelated with ones denoted in the PVO. This Chapter also presented PVO in a RDF-based knowledge representation language, such as DAML. By representing with DAML, we are able to access and manipulate PVO systematically, which leads to automatic visualization creation. In addition, this Chapter showed how to query PVO to search proper visualizations to meet user goals.

```

<!-- NAMESPACE -->
<dql-ql:query
  xmlns:dql-ql="http://www.w3.org/2003/10/dql-ql-syntax#"
  xmlns:var="http://www.w3.org/2003/10/dql-ql-variables#"
  xmlns:iw="http://www.ksl.stanford.edu/software/IW/spec/iw.daml#
  l#"
  xmlns:tkb="http://www.vrac.iastate.edu/~leekukji/pvo.daml#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:dql="http://www.w3.org/2002/07/dql#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#">
<!-- Query (X? supports interaction analysis type) -->
<dql-ql:queryPattern>
<rdf:RDF>
  <rdf:Description rdf:about=#supportAnalysisType>
    <var:x rdf:resource=#Interaction/>
  </rdf:Description>
</rdf:RDF>
</dql-ql:queryPattern>
<!-- MUST-BIND VARIABLE -->
<dql-ql:mustBindVars>
  <var:x/>
</dql-ql:mustBindVars>

<!-- ANSWER PATTERN -->
<dql-ql:answerKBPattern>
  <dql-ql:kbRef
  rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml"
  />
</dql-ql:answerKBPattern>
</dql-ql:query>

```

Figure 10. DAML representation of "What Visual Designs Support Interaction Analysis Type"


```

<owl-ql:answerBundle  xmlns:owl-ql="http://www.w3.org/2003/10/owl-ql-
syntax#"
  <!-- Namespace -->
    xmlns:var="http://www.w3.org/2003/10/owl-ql-variables#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!-- The variable x is bound with MATRIX_VIEW_DESIGN -->
  <owl-ql:answer>
    <owl-ql:binding-set>
      <var:x
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#MATRIX_V
IEW_DESIGN"/>
    </owl-ql:binding-set>
  <!-- ANSWER -->
    <owl-ql:answerPatternInstance>
      <rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#">
        <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Inter
action">
  <!-- MATRIX_VIEW_DESIGN supportAnalysis Interaction -->
    <tkb:supportAnalysis
xmlns:tkb="http://www.vrac.iastate.edu/~leekukji/pvo.daml#"
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#MA
TRIX_VIEW_DESIGN"/>
    </rdf:Description>
  </rdf:RDF>
    </owl-ql:answerPatternInstance>
  </owl-ql:answer>
  <owl-ql:continuation>
    <owl-ql:termination-token>
      <owl-ql:end/>
    </owl-ql:termination-token>
  </owl-ql:continuation>

```

Figure 11. DAML representation of the Answer of “What visual design does support interaction analysis type”

CHAPTER 4. VISUALIZATION COMPONENT

This Chapter provides more details about how to create a visualization component (VC). It is important that a VC design is consistent and systematic for its reusability; a VC should keep its syntactic and semantic meanings when integrated into another VC. First, we present different aspects of VC that we need to consider in designing a new VC. We then present a creation methodology which consists of several steps, each of which is a process of defining each aspect of the VC.

4.1 Visualization Component (VC)

A Visualization Component (VC) is a component implementation of a view that displays visual elements based on a visual design (Figure 12). A visual element could be any graphical primitive (e.g., line, bar) which visually encodes a single data or a complete view that displays a whole data set. It is important that the interfaces of VC should be well defined in a way that supports uniform access and manipulation.

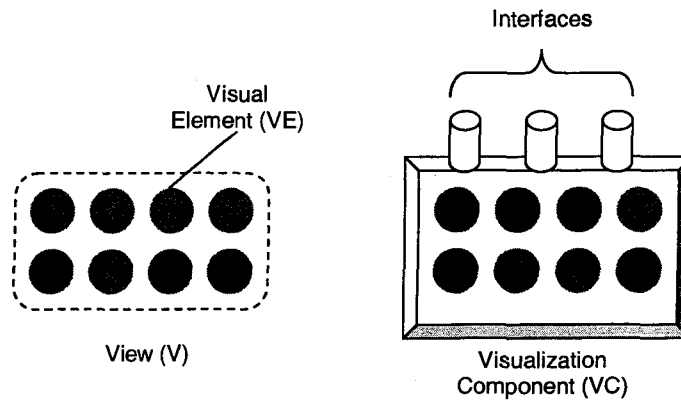


Figure 12. Visualization Component

The design of VC should address the following issues:

- Modularity: VC should be fully manageable without any dependency to other software modules
- Componentization: End user of VC should not be aware of its detail implementation
- Hierarchy: VC can contain other sub VCs.
- Composability: VC, once designed and created, can be reused to create other VCs

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* [111]. 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. For instance, the attributes of interface and component identification (e.g., location, execution platform) describe the component's concept, and the information about how to initiate, access, and interact, its content. The properties of applicable problem domains denote the component's context.

To represent components uniformly, we provide a specification for describing each concept, content, and context of a component. A specification helps developers to develop consistent interfacing mechanisms for accessing, viewing, and managing heterogeneous components. Incorporating the component descriptions of current component technologies [27], we have developed VC-specific specifications. For instance, we are capturing information about interfaces of off-the-shelf visualizations: what the most commonly used interface types are, what their parameters are, and what the common control events are. When someone develops and describes a component, he or she needs to follow the given description in order to make the component uniformly identifiable. With this approach, components may be specified uniformly and be accessible consistently. In the following methodology section, we describe our component specifications in part of VC creation.

4.2 Methodology for Creating Visualization Component

One of the main goals of our efforts is to create a structured design methodology that could be used to create a visualization component. It is important that the visualization component which is developed based on the methodology should be reusable and integratable in URV-based systems. Also the methodology needs to be easily implemented.

The creation methodology is basically a process of determining each element of view, which is defined in the previous section. Employing an object-oriented software design, the methodology consists of the following steps:

- **Step 1: Define visualization knowledge:** This step is a process of identifying the goals of VC
- **Step 2: Design a visual design:** This step is a process of designing the structure and relationship of visual objects in effective format. Design decisions such as how to layout visual elements, what supplementary labels are necessary, and where to position them, are made in this step.
- **Step 3: Determine graphical primitive(s):** This step is a process of designing graphical primitives so that they best work with the visual design defined in Step 2. It also decides how to map the data to the designed graphical primitives.
- **Step 4: Implement graphical primitive(s):** This step is a process of building graphical primitives defined in Step 3. It involves implementing an graphical primitive class which defines common attributes and operations of all graphical primitives, and by inheriting them, building specific graphical primitives.
- **(Step 5): Build a template VC:** In this step, we define common interfaces and operations for consistent manipulation and access of VC, and based on them, create a template component for specific VCs. This step is required only once. Once the

template VC is created, we can reuse it without modification for other specific VC development.

- **Step 6: Implement a specific VC:** This step is a process of building the specific VC, which reflects the visual design defined in Step 2, by specializing the template component designed in Step 5

The next sections describe each step in detail. To illustrate how the methodology works, a bar chart VC, which is useful for displaying quantitative data, is developed through the rest of this Chapter.

4.2.1 Step 1: Define Visualization Knowledge

The first step in the VC creation methodology is to define visualization knowledge. This step involves identifying the goals of the VC. The description of the goals then becomes a unique knowledge about the VC, which is used on its retrieval. The task of identifying goals involves determining the characteristics of the performance information to be visualized. In other words, this task is a process of determining the properties of performance data provided by RMC. The following questions need to be answered for identifying goals:

- What resources can be best visualized by the new VC?
- What types of performance event (or/and) metric data of the resource(s) can be best visualized by the new VC?
- What types of analysis does the new VC support?

Example: Define visualization knowledge of VC

We aim at designing a Bar chart VC, which is good for visualizing:

- Resources: CPU, Memory, Disk, Network activities
- Performance data: Metric data, such as load, utilization, bandwidth, size, etc
- Analysis type: Observing statistical status/trend of resource over time and comparison between different metric data

Based on this knowledge, we can construct the knowledge of the Bar chart VC as follows:

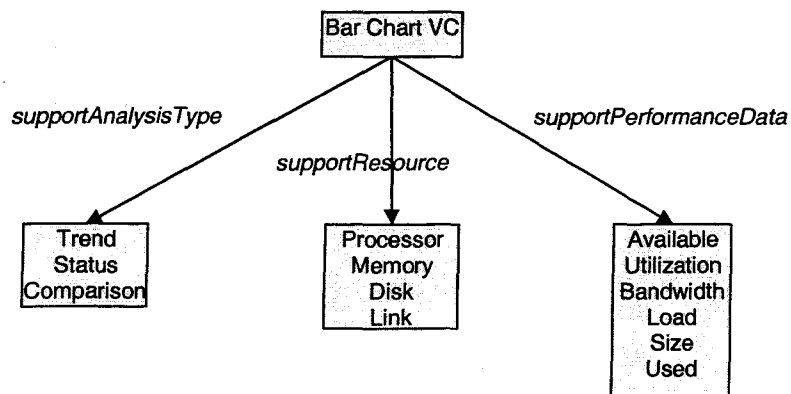


Figure 13. Knowledge of VC

The following DAML representation of this barchart can be added to the PVO.

```

<rdf:Description rdf:about="#BarChartInstance">
  <rdf:type>
    <!-- Instance of BAR_CHART_DESIGN -->
    <daml:Class rdf:about="#BAR_CHART_DESIGN" />
  </rdf:type>
  <!-- Supporting analysis types -->
  <ns0:supportAnalysisType rdf:resource="#Trend" />

```

```
<ns0:supportAnalysisType rdf:resource="#Status"/>

<!-- Supporting performance data types -->
<ns0:supportPerformanceData rdf:resource="#Processor"/>
<ns0:supportPerformanceData rdf:resource="#Memory"/>

<!-- Supporting resource types -->
<ns0:supportResource rdf:resource="#Available"/>
<ns0:supportResource rdf:resource="#Utilization"/>
<ns0:supportResource rdf:resource="#Bandwidth"/>
<ns0:supportResource rdf:resource="#Load"/>

</rdf:Description>
```

4.2.2 Step 2: Design a Visual Design

Visual design is very diverse with rich semantic content. It includes both graphical and stylistic content. This step involves determining their structure and relationship (i.e. style) and extra graphical primitives that support the visual design. Text labels, legends, and axis are examples of the extra supporting graphical primitives. Graphical primitives can be specified using a list. Style, however, are often less precise, which makes it hard to develop a consistent representation. Representation of visual design is out of the scope of this research. This research focuses on better uses of visual design depending on performance problems. Several visual design representation works can be found in [26][63][70][98].

Example: Define visual design

The basic design of bar chart is to layout given visual elements horizontally along with x-axis. Y-axis represents data values of visual elements. In addition, we need labels to describe a type of each axis and a title of VC.

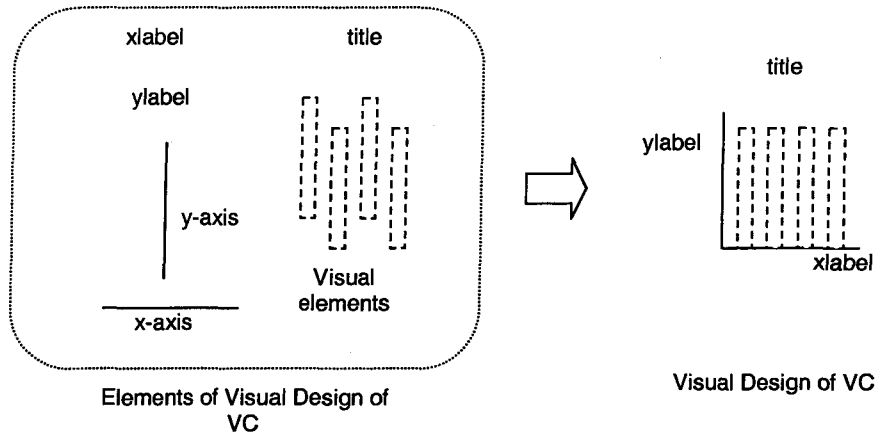


Figure 14. Elements of Bar Chart visual design

4.2.3 Step 3: Determine Graphical Primitive

Visual encoding is a process of determining a graphical primitive; it involves selecting a proper visual object and determining its attribute used for encoding the data. Line, marker, bar, and polygon are good examples of visual object. Length, color, pattern, width, and size are examples of the attributes of visual object. Graphical primitives are generally easier to define than stylistic qualities. That is because there is a finite set of accepted techniques and taxonomies for effective information visualization [70][101][91].

Table 4. Effectiveness rankings of different perceptual encoding techniques for three basic data types (Adapted from [70])

<i>Quantitative</i>	<i>Ordinal</i>	<i>Nominal</i>	
Position	Position	Position	most effective
Length	Color Intensity	Color Hue	↑
Angle	Color Hue	Pattern/Texture	
Slope	Pattern/Texture	Connection	
Area/Volume	Text	Containment	
Color Intensity	Connection	Shape	
Text	Containment	Text	
	Length	Length	
	Angle	Angle	
	Slope	Slope	↓
	Area/Volume		least effective

This step involves identifying visual objects that best work with the visual design defined in the previous step and determining its effective attribute. Table 4 shows one of previous studies about the effectiveness of each attribute for different data type [70]. Although it can not be used as an absolute rule, it helps to decide a better attribute for effective use of the visual object. By applying this effectiveness study to visual objects, we have come up with Table 5 which shows a preferred data type depending on each visual object and its attribute. Further, By integrating Table 4 and 5, Table 6 summarizes effectiveness rankings of graphical primitive for each data type.

Table 5. Preferred data type for each graphical primitive and its attribute

<i>Graphical Primitive</i>	<i>Attribute</i>	<i>Preferred data type</i>
Line	Length	Quantitative
	Color	Ordinal
	Pattern	Nominal
	Width	Quantitative
Marker	Position	Quantitative, Ordinal
	Size	Quantitative
	Color	Ordinal
	Pattern	Nominal
	Shape	Nominal
Bar	Height	Quantitative

<i>Graphical Primitive</i>	<i>Attribute</i>	<i>Preferred data type</i>
	Color	Ordinal
	Pattern	Nominal
Area	Size(Angle)	Quantitative
	Color	Ordinal
	Pattern	Nominal

Table 6. Effectiveness rankings of graphical primitive for each data type

<i>Data type</i>	<i>Graphical Primitive + attribute</i>	<i>Effectiveness</i>
Quantitative	Marker position	Most
	Line length	↑
	Area size	
	Marker size	
	Line width	↓
	Bar height	Least
Ordinal	Marker position	Most
	Line color	↑
	Marker color	
	Bar color	↓
	Area color	Least
Nominal	Line pattern	Most
	Marker pattern	↑
	Marker shape	
	Bar pattern	↓
	Area pattern	Least

This step can be recognized as a guideline for selecting appropriate graphical primitives. The actual decision could vary depending on each person's preference.

Example: Determine graphical primitives

Let us pick a bar and its height as a graphical primitive to present quantitative data.

4.2.4 Step 4: Implement Graphical Primitives

This step involves implementing a graphical primitive class which defines common attributes and operations of all types of graphical primitives. This is one time process

such that other specialized graphical primitives (e.g., bar, marker) can be constructed as a sub class of the graphical primitive class. Having a graphical primitive class supports consistent access & manipulation of various graphical primitives.

Figure 15 shows a class hierarchy in Unified Modeling Language (UML) [116] class diagram. The highest class, *graphical component class*, includes everything from providing layout hints to supporting painting and events. It also supports for adding components to the container and laying them out. *JComponent* class in Java is a good example. Inheriting such a class help to avoid detail low-level implementation of graphics operations (e.g., double buffering, painting, refreshing).

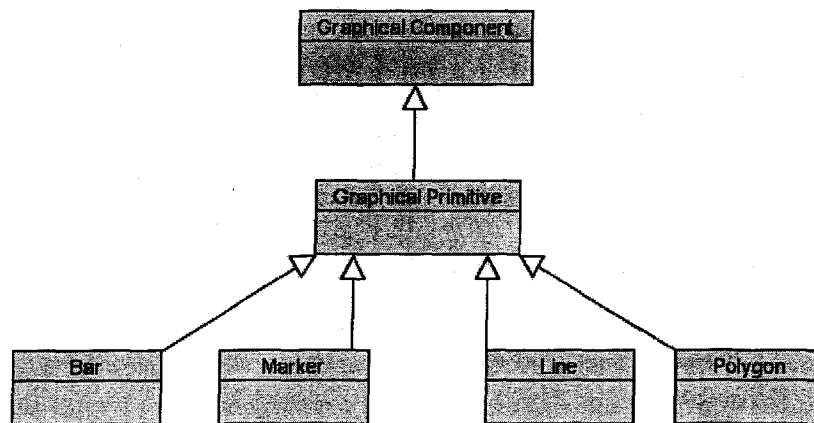


Figure 15. Class hierarchy of graphical primitive

The design process of the graphical primitive class involves:

- Identifying class attributes: It defines a list of common variables/information that a graphical primitive must contain
- Defining class constructors: It defines different ways of constructing classes

- Identifying class operations: It defines a list of common operations of graphical primitive.

In this research, we have defined the following class information (Figure 16) which can be extended and/or reused. Figure 17 shows a pseudo code for graphical primitive class.

Graphical Primitive Class

- Attributes

- timeStamp: As a graphical primitive represents a single performance data, the time stamp value, that all performance data have, should be presented
- colorValue: Every graphical primitive may or should have data which can be encoded with its color
- value1, value2, value3: data values represented by this graphical primitive

- Constructors

- GraphicalPrimitive (): an empty class creation
- GraphicalPrimitive (timeStamp, value): creation with data values
- GraphicalPrimitive (GraphicalPrimitive gp): creation with another graphical primitive class. In particular, this constructor is used on converting a type of graphical primitive. For

instance, a height of bar object is compatible with a position of marker object in a sense that both graphical primitive supports encoding quantitative data.

- Operations

- `encode (value)`: It gets a data value and encodes it with supporting visual attribute. For instance, a bar encodes primarily a data value with its height. If a new data needs to be added to the same bar, its value is then encoded with the bar width. The sequence of preferred encoding is based on Table 5.
- `setGPsize (width, height)`: This operation allows resizing a graphical primitive. It does not change actual data value encoded with the graphical primitive.
- `getPreferredSize ()`: This operation returns its preferred size when displayed on a screen. VC determines its preferred size by summing up the preferred sizes of all graphical primitives.
- `duplicate ()`: It returns a new cloned graphical primitive.

Figure 16. Graphical primitive class definition

```

Class GraphicalPrimitive extends Graphical Component {
  //Attributes
  timeStamp
  colorValue
  value1, value2, value3

  //Creation
  GraphicalPrimitive() {}
  GraphicalPrimitive(timeStamp, data) {}
  GraphicalPrimitive(GraphicalPrimitive gp) {}

  //Encode data
  Encode(data)

  //Duplicate
  Duplicate()

  //Set a size
  setGPSize(width, height)

  //Get a preferred size
  getPreferredSize()
}

```

Figure 17. Graphical primitive pseudo code

Then the design of each specific graphical primitive can be done by:

- Identifying a graphical primitive specific encoding attributes: It adds new attributes specific to the graphical primitive
- Redefining encoding operations: It defines a sequence of encoding preference.
- Implementing a visual representation of the specific graphical primitive: It actually implements the look of the graphical primitive using graphics methods (e.g., draw a rectangle, line,) supported by *Graphical Component class*

Example: Determine graphical primitives

The bar graphical primitive can be constructed by extending the graphical primitive class. In case of 'Bar', we set a encoding preference as height → Width → Color. In other words, if a single bar presents more than one value, it shows a primary value with its height, a second value with its width, and so on.

```

Class Bar extends GraphicalPrimitive {
    //Define additional attributes

    Assign value1 to the height;
    Assign value2 to the width;
    Assign value3 to the color;

    //operations

    //Creation
    Bar() {}
    Bar(timestamp, data) {...}
    Bar(GraphicalPrimitive gp) {...}

    //Redefining operations if necessary
    Encode(data)

    //Duplicate
    Duplicate()

    //Set a size
    setGPSize(width, height);

    //Get a preferred size
    getPreferredSize();

    paint(){
        Draw rectangle with height, width, and color;
    }
}

```


4.2.5 Step 5: Implement a Template Visualization Component

This step is a process of defining common attributes, operations, and interfaces for consistent manipulation and access of VC, and based on them, creating a template VC. The goal of the template VC is to facilitate creation of specific VCs by reusing most of VC properties. Figure 18 shows a class hierarchy of VC and its related classes.

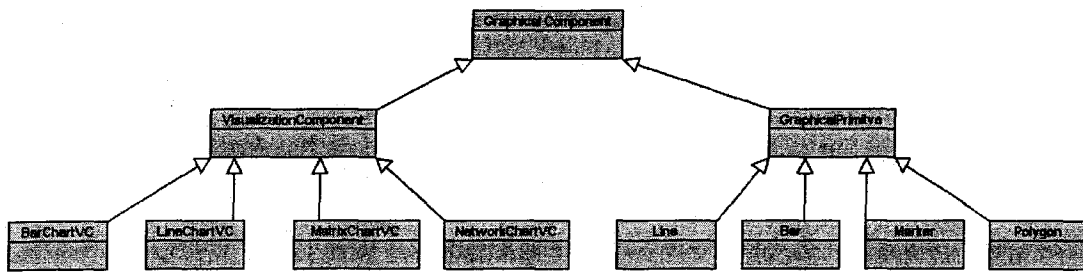


Figure 18. Class hierarchy of visualization component

Like *GraphicalPrimitive* class, a visualization component class also inherits, *graphical component* class, which enables a consistent interchange of visualized information (e.g., graphical primitives, VC).

We have defined the following attributes and operations to meet the requirement of VC, which is discussed in Section 4.1.

Template Visualization Component Class

- Attributes

- o `visualDesignName`: name of visual design that the VC presents
- o `startTimeStamp`: start collection time of data that are presented in VC
- o `endTimeStamp`: end collection time of data that are presented in VC
- o `currTimeStamp`: time of data that are currently displayed in VC
- o `visualElementList`: a list of visual elements.
- o `nodeInfo`: a list of nodes that are presented in VC
- o `resourceInfo`: a list of resources that are presented in VC
- o `perfDataInfo`: a list of performance data types that are presented in VC

- Constructors

- o `VisualizationComponent ()`: an empty class creation
- o `VisualizationComponent (visualElementList)`: creation with a list of visual elements
- o `VisualizationComponent (VisualizationComponent vc)`: creation with another VC. In particular, this constructor is used for converting one VC to its compatible VC. For instance, a bar chart VC can be compatible with a line chart VC in a sense that both VC supports quantitative performance metric data

- Operations

- o `buildView()` : it implements a visual design. In other words, this operation layouts the visual elements based on the visual design.
- o `getMaxHeightValue()` : it returns a max height value (in display) of VC
- o `getMaxWidthValue()` : it returns a max height value (in display) of VC
- o `adjustComponentSize()` : it adjusts the size of VC and its sub components (e.g., visual elements).
- o `setMargins ()` : it sets margin values of VC
- o `paint ()` : it draws extra graphical primitives, such as text labels, x-axis, and y-axis.
- o `duplicate ()` : it clones the VC

Figure 19. Visualization component class definition

In addition, the following interfaces need to be implemented for manipulation and access of VC by other components.

```
Interface VC_Interfaces {

    set/getVisualDesignName ();
    set/getAnalysisType ();
    set/getNodeInfo();
    set/getResourceInfo();
    set/getPerfDataInfo();

    addVisualElement (GraphicalComponent gc);
    addVisualElement (GraphicalPrimitive gp);
    addVisualElement (VisualizationComponent vc);
    set/getVisualElementList (visualElementList);

    set/getCurrTimeStamp();
}
```

```

getStartTimeStamp();
getEndTimeStamp();

    setVCsize(width, height); // Set a display size of VC
}

```

Figure 20. Visualization component interfaces

4.2.6 Step 6: Implement a Visualization Component

Inheriting a template VC, this step involves defining additional attributes and redefining the operations. In particular, implementation of visual design requires redefining a `buildView()` operation for lay-outing visual elements and a `paint()` operation for drawing supplementary graphic objects (e.g., labels, axis).

```

Class VC extends TemplateVC {

    //defining additional attributes if necessary
    //redefining operations if necessary
    //Implementing visual design

    //redefining buildView()
    buildView() {

        // Based on a visual design, layout visual elements
        For (all visual elements) {
            add a visual element;
            set the location of the visual element;
        }
    }

    //redefining paint()
    paint() {

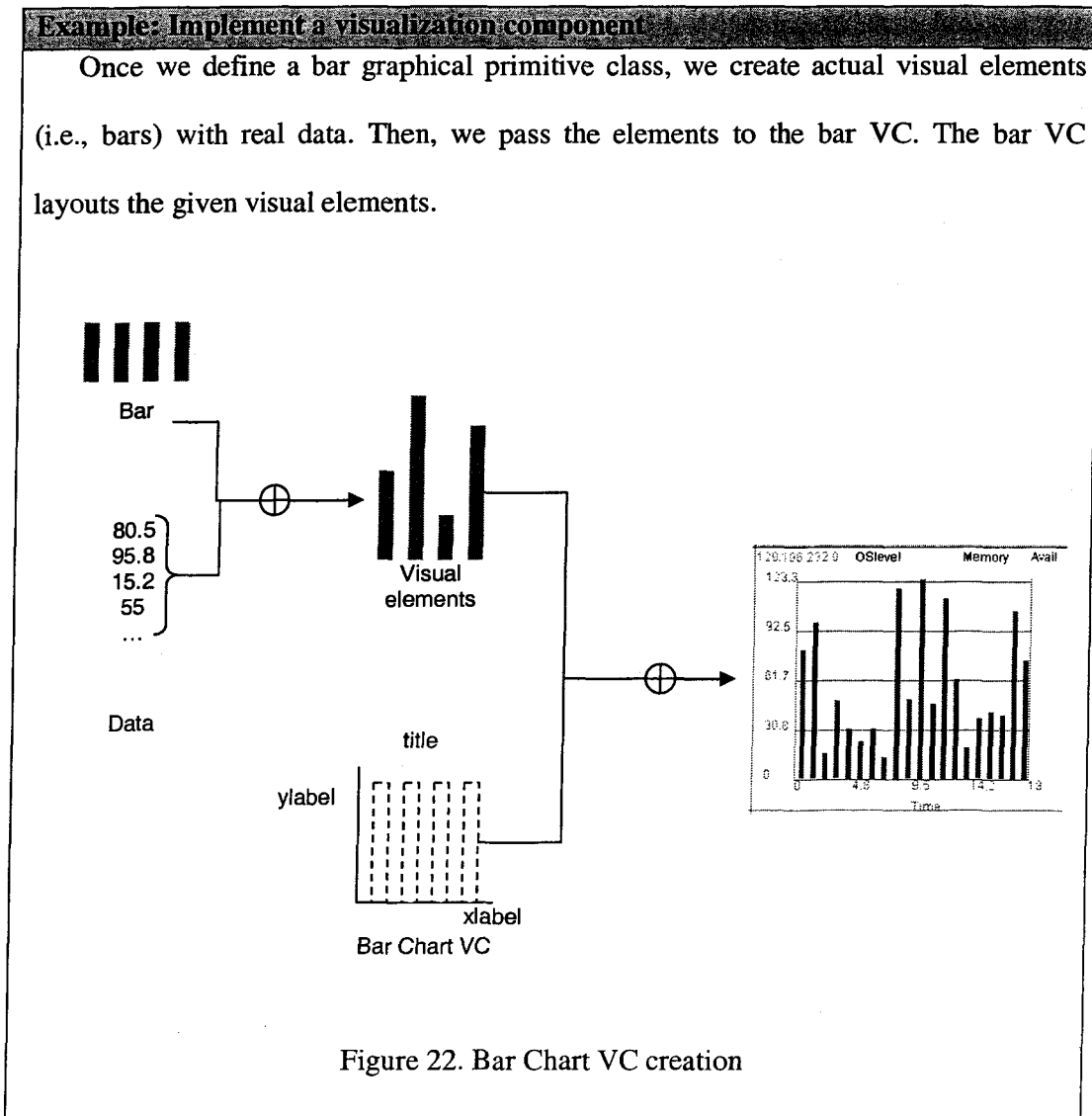
        Draw labels (e.g., x label, y label, title);
        Draw supplementary visual objects (e.g., x, y axis);
    }
}

```

Figure 21. Pseudo code of Visualization Component (VC)

Figure 21 shows a pseudo code of VC. A specific VC extends a template VC by redefining attributes and/or operations if necessary.

Once we define the VC, we add it to the PVO as a new instance of the selected visual design.



4.3 Summary

This Chapter provided the definition of visualization component, including the requirement of building composable and sharable visualization components. In particular, we provided a set of interfaces and data structures of visualization components for seamless information integration and uniform manipulation of the components. Further, we presented a visualization component creation methodology, which consists of multiple steps: visualization knowledge definition, visual design definition, graphical primitive determination, graphical primitive implementation, template VC creation, and VC specialization. The section 4.2 demonstrated each step with the example of creating a bar chart visualization component.

CHAPTER 5. VISUALIZATION COMPOSITION

The ultimate goal of this research is to provide a systematic creation method of system-level performance visualizations. This Chapter describes different types of visualization compositions and a composition methodology.

5.1 Visualization Composition

Composition simplifies view creation for typical users, not unlike the use of a chart wizard in a desktop spreadsheet tool. Visualization composition is for creating a System-level Performance Visualization (SPV) that presents data and/or visual correlation between multiple resources. Here, correlation does not necessarily imply statistical correlation. Instead, it presents perceptually inferred new information that can be obtained through a new composite visualization. Data correlation presents the degree of association between performance data of resources. For instance, a scatter plot, which shows ‘association between CPU load and network latency’, is an example of data-correlated SPV. In data-correlated SPVs, the association is directly visible. On the other hand, a visual-correlated SPV presents association indirectly. It rather focuses on integration of existing visual representations. For instance, a network graph, of which nodes themselves are pie charts showing memory used/avail of hosts, is a visual-correlated SPV, which is created by composing a network graph and several pie charts. In general, data-correlated SPVs are limited to a few specific visualizations (e.g., a scatter plot), of which views are fixed. In this research, we focus on visual correlation approach of SPV.

Depending on the level at which visualization composition happens, we can have different composite views. We have categorized three different levels: component-level, graphical primitive-level, and view-level. The purpose of this categorization is to help users in choosing an appropriate composition method for goals.

Component-level composition enables users to construct custom exploration interfaces by coordinating multiple views [77]. The main goal of this approach is to provide consistent methods to explore multiple visualizations concurrently. For instance, a system manager wants to analyze a web log. Using multiple coordinated visualizations, the manager can explore various aspects of the log. A geographical map view displays where HTTP requests came from. As the manager selects a particular location on the map, a table view shows a list of pages requested from the selected location. At the same time, a bar chart view shows the frequency of each requested page (Figure 23). This composition assumes that a visualization developer must know in advance which visualizations should be coordinated; no guidance is provided.

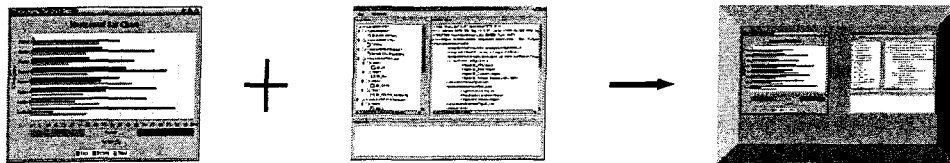


Figure 23. Component-level visualization composition

Graphical primitive-level composition composes views based on the graphical primitives that encode the same information, and synchronizes the views based on a common independent variable. It results in a new view that synchronizes/joins (spatially and temporally) individual views into a single graph based on a common independent

variable. Synchronization between two views depends on a concrete set of transformation rules; it is straightforward given design knowledge about the visualization components. 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. A composition operation at graphical primitive level is *merging*. The goal of merging is often to compare two views and/or identify correlation between the views.

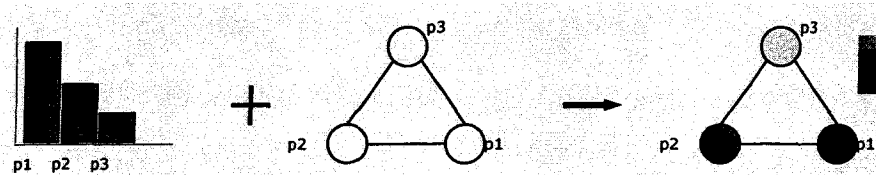
There are three commonly used merging operations [102]:

- **Merging by union:** it combines graphical primitives of views using set union. Suppose there are two bar charts: one shows available memory over time with executing a particular program and the other shows the same without executing. By merging data sets of two bar charts, we can come up with a new bar chart that shows both information in a single view, which leads to the estimation of memory requirements over time for the program (Figure 24 (a)).
- **Merging by encoding:** It merges graphical primitives of views by encoding each visual element of one view with a compatible set of marks of the other. This composition is based on a principle that if one visual design can compromise another visual design, a composite view can be created by re-encoding graphical primitives of the visualizations [26][70][98]. Figure 24 (b) shows an example of merging-encoding composition. The bar chart view presents the utilization of three processes ($p1$, $p2$, $p3$). The network topology view shows the interaction relationship between processes; e.g., $p1$ interacts with $p2$ and $p3$. We then can compose two views by re-encoding each bar of the bar chart with the different color of each node in the network view. The new composite view shows the interaction relationship and the utilization values over time.

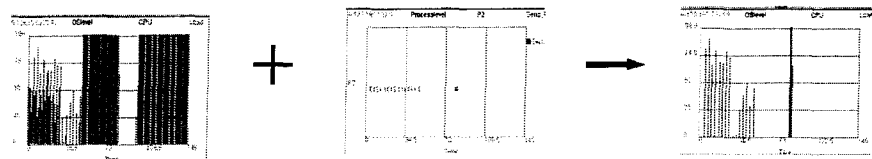
- Merging by intersection:** It combines a pair of graphical primitives by first computing their intersection, then superimposing the intersection onto one of the view. Suppose one view shows a CPU load and the other view shows a message sent event. By intersecting two views, we can identify how each sent message correlates with the CPU load (Figure 24 (c))



(a) An example of merge-union



(b) An example of merge-encoding



(c) An example of merge-intersection

Figure 24. Graphical primitive level visualization composition

View-level composition aids in the design of new visualization layers that represent the cooperation between/among resources. The operation of this type of composition is referred to as *synthesis*. It is a conscious activity by a designer that may not be automated

as fully as union. It synthesizes the attributes of visualizations and creates a new visualization that supports synthesized attributes of visualization. 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. The goal of synthesis is to enhance visual relationships among views and/or present more global information.

Suppose a matrix view and a bar chart view have been known effective for presenting *interaction* and *status* of resources, respectively, which is knowledge. In Figure 25, the matrix view presents communication bandwidths between processes, and the bar view, a utilization of each process. Both views have different visual designs and neither of them can compromise the other. We then need to create a new visualization layer to compose these views. A new visualization should accommodate both visualization knowledge (i.e., interaction and status). A network view can present both design knowledge in the sense that interaction and status are represented by the edges and different color of the nodes, respectively, respectively.

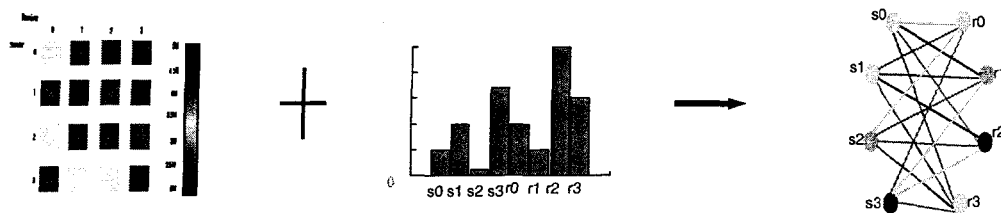


Figure 25. View-level visualization composition

5.2 Methodology for View-Level Composition (Synthesis)

The motivation for composition in all cases is analyzing the performance of multiple resources. In particular, URV with PVO provides a way to look up appropriate visualizations to display particular performance problems. The ultimate goal of our efforts is to develop a structured design methodology for creating a system-level performance visualization, focusing on employing visualization knowledge. This section describes our methodology on view-level composition operation, synthesis.

The methodology consists of the following steps:

- **Step 1: Compose knowledge of visualization components:** This step is a process of composing different knowledge of VCs and create an integrated knowledge that a new composite VC will present
- **Step 2: Identify new visualization component(s):** This step involves searching visualization component(s) that matches the integrated knowledge. It first constructs a query and looks up into Performance Visualization Ontology (PVO) for matching
- **Step 3: Construct a new VC:** This step is a process of specializing the found VCs with the visual elements of source views

5.2.1 Step 1: Compose Knowledge

The first step in the synthesis methodology is to merge visualization knowledge. By merging knowledge, we can keep the same semantics in the final composite VC. As described in Chapter 4, each performance visualization knowledge can be a combination of supporting analysis types (A), resource types (R), and performance data types (P).

The knowledge of $VC1$: $(A1, R1, P1)$, The knowledge of $VC2$: $(A2, R2, P2)$

Compose: $(A1, R1, P1) \oplus (A2, R2, P2)$

$\rightarrow (\{A1, A2\}, \{R1, R2\}, \{P1, P2\})$

A : a set of analysis types, R : a set of resource type, P : a set of performance data types, \oplus : composition

Figure 26. Merging knowledge

Given two visualization components, $VC1$ and $VC2$, their knowledge composition can be done by merging each element of knowledge. Figure 26 describes a knowledge merging process.

Example: Compose Knowledge

Suppose there are three VCs: two displays message send/receive events of sender and receiver, and the other displays status of processes involving this message transfer.

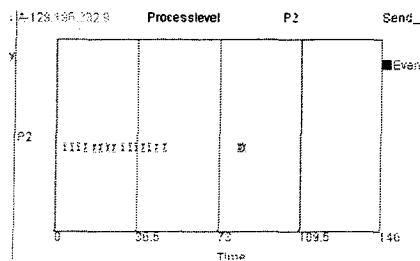


Figure 27. $VC1$ (displays the occurrences of send events with markers over time)

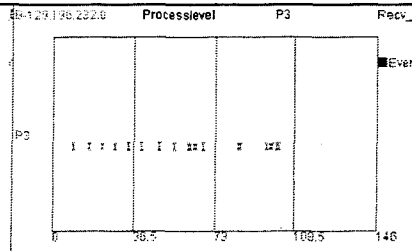


Figure 28 VC2 (displays the occurrences of receive events with markers over time)

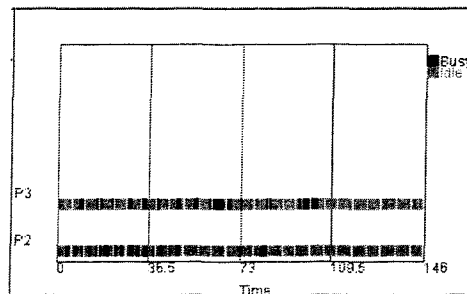


Figure 29. VC3 (displays the status of processes with different colors (yellow: idle, green: busy))

Knowledge of VC1 = ({Status}, {Process}, {Send})

Knowledge of VC2 = ({Status}, {Process}, {Receive})

Knowledge of VC3 = ({Status}, {Process}, {Busy, Idle})

A new knowledge can be obtained by merging them: ({Status}, {Process}, {Busy, Idle, Send, Receive}).

5.2.2 Step 2: Identify New Visualization Component(s)

In this step, using the newly composed knowledge in the previous step, we query Performance Visualization Ontology (PVO) to find new visual designs and VC instances

that match. As denoted in Section 4.4, we use a DAML query language to construct a query.

Example: Identify New Visual Component(s)

In this step, we construct a DQL query for asking whether/which visualization component(s) support visualization of ({Status}, {Process}, {Busy, Idle, Send, Receive}).

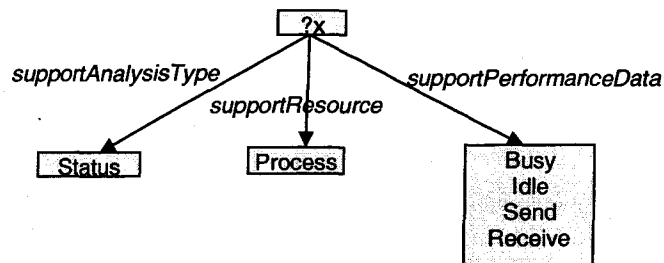


Figure 30. A RDF-based query model of the composed knowledge

Reasoning the query, the PVO query system identifies a matched visual design, which is a space diagram design. In particular, a certain instance VC of space-diagram design supports ({Status, Interaction}, {Process}, {Busy, Idle, Send, Receive}), which accommodate the all elements of queried knowledge. Note that the new VC also supports “interaction” analysis.

5.2.3 Step 3: Construct a New Visualization Component (VC)

This step involves specializing the newly found VCs with the visual elements of the source VCs. This process makes use of `addVisualElementList` (`visualElementList`) method, which enables adding existing visual elements to

the VC. Then, the new VC builds a view with the added visual elements based on its visual design.

Example: Construct a New VC

In this step, we specialize a newly found VC (SpaceDiagram).

```
VisualizationComponent vc = new VisualizationComponent (SpaceDiagram);
vc.addVisualElementList (vc1.getVisualElementList());
//add VC1's visual elements
vc.addVisualElementList (vc2.getVisualElementList());
//add VC2's visual elements
vc.addVisualElementList (vc3.getVisualElementList());
//add VC3's visual elements
vc.buildView(); //build a view
```

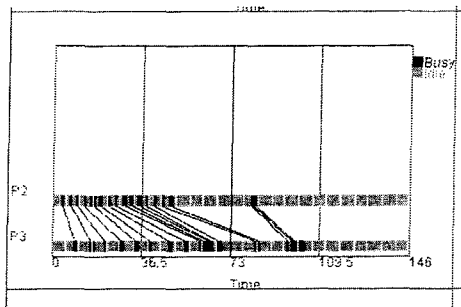


Figure 31. A new composite VC

As shown in Figure 31, all visual elements of source VCs are in the new VC, but now we also have new graphical primitives, which are links connecting two events (send, receive). These graphical primitives are in part of the visual design of the new VC.

5.3 Summary

This Chapter provided the principle and methodology of the proposed visualization composition approach. First, we categorized different levels of composition to help users determine an appropriate composition method to meet the user goals. The levels include a component-level, graphical primitive-level, and a view-level. Further, we defined a set of composition operations for graphical primitive level (merge by union, encoding, and intersection) and view-level (synthesis) compositions. In particular, this Chapter presented a methodology for synthesis, which is knowledge-based composition. The methodology consists of three steps: knowledge composition, visualization search by the composed knowledge, and a final VC creation.

CHAPTER 6. SPREADSHEET FOR SYSTEM-LEVEL PERFORMANCE VISUALIZATION

This Chapter describes our new performance visualization technique that employs a spreadsheet metaphor. This technique facilitates applying our methodologies to visualization components.

6.1 Spreadsheet Visualization

Spreadsheet visualization [25][69][118] provides a robust and intuitive technique for presenting and interacting with multiple visualizations. Among its advantages, it supports many of the classic visualization methods articulated by Edward Tufte, such as small multiples, layering and separation, relational graphics, etc. For instance, Spreadsheet for Information Visualization (SIV) [25] arranges visualizations in a spreadsheet grid and maintains common viewing parameters along rows and columns of visualizations. SIV also supports operations, such as rotation or data filtering, on a group of visualizations. Such operations are effective for comparing two or more related data sets. There have been a number of extensions to SIV [69][118] and spreadsheet visualization in general, as well as applications of spreadsheet visualization to numerous information domains. Most spreadsheet visualizations are based on homogeneous data sets. The operations on cells (data objects or views) are limited to numeric computations, which cannot deal with the semantics of data sets. A tool, such as VisAD [55], uses data and display models with component metadata that is more flexible; however, it is specific to scientific

visualization. New spreadsheet operations are needed to support integration of performance data at multiple levels based on their semantics.

6.2 Spreadsheet for System-Level Performance Visualization

In this research, we extend the SIV approach to the performance visualization domain, developing a tool called Spreadsheet for System-Level Performance Visualization (SSPV) to enable system-level analysis of complex computing systems by integrating performance information. This approach will not only result in a usable tool, but it also will provide a means to develop and demonstrate URV services and operations, such as composition. Applying the SIV approach to performance visualization provides the following benefits:

- **Ability to compare performance over multiple time periods:** Comparing resource behaviors between different time periods is very useful to drilldown performance issues. For instance, given the last week data, which could be a baseline, we can compare this week data to see any major difference. If there is a significant performance degradation, we can start drilldown the problem by comparing, for instance, different network setups, and/or memory allocation policies of the machine. The SIV provides a greater flexibility in placing multiple visualization, each of which presents different time periods.
- **Ability to accommodate a large set of visualizations:** As we discussed in Chapter 1, monitoring a distributed application requires monitoring many related resources. In that perspective, the SIV is scalable to accommodate many visualizations in one place.

Figure 32 shows the snapshot of SSPV, in which a cell contains a visualization component (VC). SSPV consists of three main parts: toolbar, worksheet, and time control. The toolbar supports several basic spreadsheet operations, such as add/remove columns/rows/worksheet, and copy/paste/select cells. The primary functionality of SSPV is to host visualization components. After the creation of VC, we can place the VC in any cell in the worksheet area. The time control allows users observe data at a specific time. Once we select visualization component(s), we can move the time control bar at the specific time. Then, the visualization components only display data at the specified time. This functionality is useful when observing correlation between multiple resources over time.

SSPV supports composition operations: merge and synthesis on VCs. In particular, compared with other SIV-based tool, a key innovation of SSPV is to support synthesis operation. Knowledge operations use knowledge to manipulate and transform the information at higher levels and/or with more advanced analysis, such as correlation of data sets. Based on visualization knowledge, synthesis operation can result in a new visualization that integrates performance information across the selected resources, levels, or aspects.

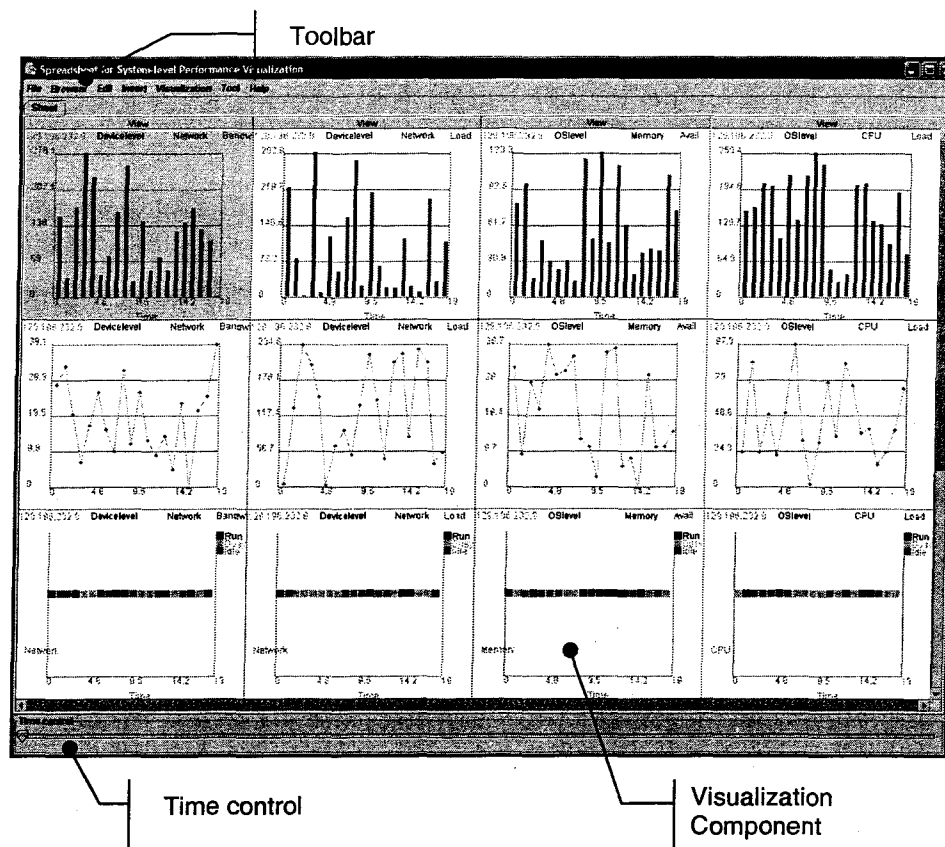


Figure 32. System-Level Performance Visualization (SSPV)

To facilitate a connection to RMCs, SSPV has its own RMC directory browser (Figure 33), which is invoked from Browser menu on the toolbar. The directory browser connects to a directory service, which contains a list of registered RMCs, and displays the RMC information.

Name	Target node	Level	Resource	Metric	Service	Port	Method	Location	Select
CPU_B_Load	B-129.196.232.9	OSlevel	CPU	Load	DemoRMC	DemoRMCPRC	get_cpu_b_load	http://localhost:8080/D	<input type="checkbox"/>
CPU_A_Load	A-129.196.232.9	OSlevel	CPU	Load	DemoRMC	DemoRMCPRC	get_cpu_a_load	http://localhost:8080/D	<input type="checkbox"/>
P3_Run_Status	B-129.196.232.9	Processlevel	P3	Status	DemoRMC	DemoRMCPRC	get_p3_run_st	http://localhost:8080/D	<input type="checkbox"/>
P2_Run_Status	A-129.196.232.9	Processlevel	P2	Status	DemoRMC	DemoRMCPRC	get_p2_run_st	http://localhost:8080/D	<input type="checkbox"/>
P3_Recv_Event	B-129.196.232.9	Processlevel	P3	Recv_Event	DemoRMC	DemoRMCPRC	get_p3_recv_e	http://localhost:8080/D	<input type="checkbox"/>
P2_Send_Event	A-129.196.232.9	Processlevel	P2	Send_Event	DemoRMC	DemoRMCPRC	get_p2_send	http://localhost:8080/D	<input type="checkbox"/>
A_Memory_Load	A-129.196.232.9	OSlevel	Memory	Load	DemoRMC	DemoRMCPRC	get_memory_l	http://localhost:8080/D	<input type="checkbox"/>
Retrans_AC	AC-129.196.232.9	Networklevel	Link	Retransmission	DemoRMC	DemoRMCPRC	get_retrans_AC	http://localhost:8080/D	<input type="checkbox"/>
Retrans_AB	AB-129.196.232.9	Networklevel	Link	Retransmission	DemoRMC	DemoRMCPRC	get_retrans_AB	http://localhost:8080/D	<input type="checkbox"/>

Figure 33. RMC directory browser

A user can select RMC(s) of interest and create VC(s) that match with the selected RMCs by clicking **Insert Selected RMCs**. SSPV queries possible VC matches to Performance Visualization Ontology (PVO). The possible matches are suggested to the user via the preview window (Figure 34). In the preview window, the user can select one of possible matches and preview a final VC look by clicking **Preview** button. Once a VC is decided based on the user preference, the user can insert the VC into the current worksheet or create a new worksheet only with the VC.

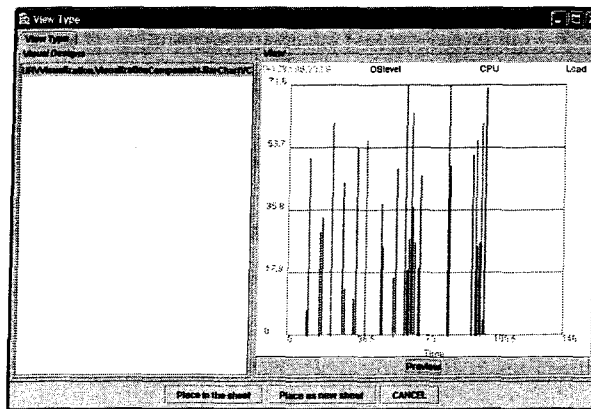


Figure 34. Matched VC and its preview dialog

The current SSPV is primarily for demonstration purpose, focusing on composition operations. It, however, could be extended to support other SIV operations, such as

filtering and numeric operations (e.g., min, max, subtract, add data), as well as more sophisticated spreadsheet operations, such as sorting (e.g., sort by a resource name).

6.3 Summary

This Chapter presents a new spreadsheet-based visualization tool, called Spreadsheet for System-level Performance Visualization (SSPV), which provides a robust and flexible visualization environment for dynamic creation of system-level visualizations. SSPV extends Spreadsheet in Information Visualization (SIV) in a way that supports performance data and composition operations (merge and synthesis). In addition, SSPV supports basic spreadsheet operations, such as add/remove rows/columns/worksheets/cells, and has its own built-in RMC directory browser

CHAPTER 7. PUTTING IT ALL TOGETHER

This Chapter describes the URV framework in action. First, we describe a performance visualization system built based on our methodologies. We use this visualization system to demonstrate an end-to-end performance monitoring scenario. The methodologies are implemented as web services which can be accessible by other software agents. Secondly, we describe a testbed, which is based on the performance scenario discussed in Chapter 1. We have implemented the testbed with a simulation package. The rest of this Chapter shows how the proposed URV approach helps drill down a root cause of performance problem.

7.1 URV System

A combination of web services and ontology technologies has a potential to provide a consistent, seamless, and intelligent integration of heterogeneous resources. Web services provide an interoperable interfacing method between heterogeneous software components. Ontology allows sharing domain specific knowledge in a consistent way, which leads to reuse of well-recognized solutions. Uniformity in URV is implemented by wrapping instrumentations with web services and by defining interfaces of visualizations. These facilities provide users with uniform interfaces for accessing, viewing, and managing heterogeneous resources. Reusability in URV is provided via performance visualization ontology. Users can retrieve existing visualization knowledge of interest and using the knowledge, they can identify the actual visualization component. Composition refers to support for higher level, e.g., system-level visualization.

Figure 35 shows a URV-based performance visualization system. A system consists of resource-monitoring components (RMC), visualization components (VC), a user application, web services, and performance visualization ontology.

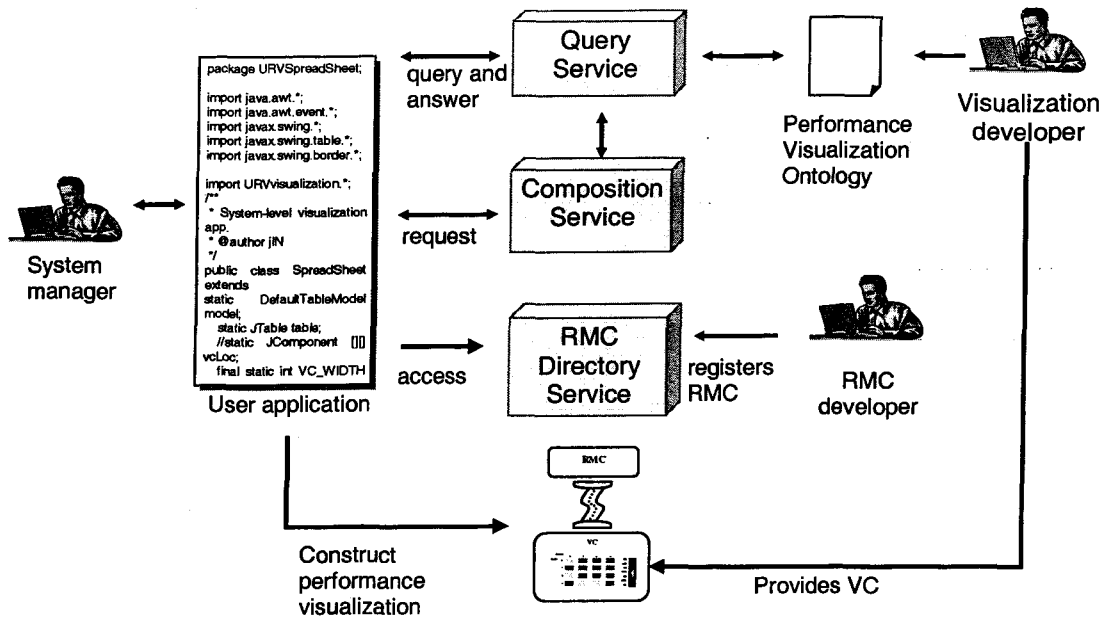


Figure 35. URV-based performance visualization system

A user application is software that makes use of URV services. It could be any software that needs performance visualization capability. In our research, SSPV, which is described in Chapter 6, is recognized as a user application that makes use of URV services. Web service based directory service can be accessed from anywhere with various programming languages. In addition, the registered information can be retrieved with any web service enabled software module. A query service enables querying-answering knowledge. Given the information about a RMC of interest, a query service returns matching visualization component(s). Given multiple RMCs and visualization

components, a composition service validates any combination of the attributes by querying performance visualization knowledge.

7.1.1 URV Services

URV supports services for reusing visualization components, as well as composing visualization knowledge to create system-level views. In this section, we describe URV services and the current approaches. Distinctive web services in URV include: (1) query service to query proper visualization knowledge, (2) directory service to search the sources of performance data (i.e., RMC) ,and (3) composition service to validate a new visualization based on multi-level, composable, reusable, and distributed components. We use Java API, JAX-RPC, to implement the web services.

A query service enables querying-answering performance visualization knowledge. For instance, a query service is able to process and answer a query like ‘What visual designs are appropriate for visualizing metric X on resource Y?’. A query service is designed as a web service to allow users or software to make a query using a designated web interface or a SOAP protocol. A DQL query pattern contains a set of DAML+OIL sentences in which some literals and/or URIs have been replaced by variables. A query answer provides bindings of terms to some of these variables. Each binding in a query answer is a URI or a literal that either explicitly occurs as a term in the knowledge. That is, DQL is designed for answering queries of the form "What URIs and literals from the knowledge denote objects that make the query pattern true?" Figure 36 shows a basic operation of the query service.

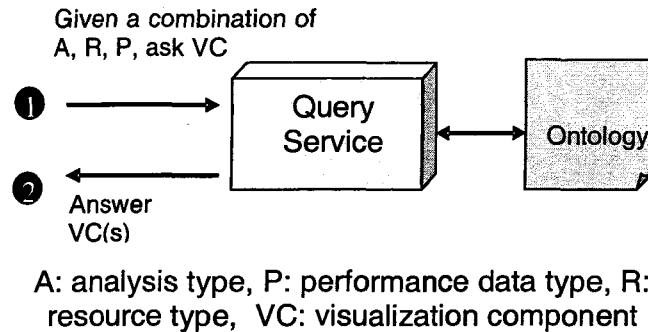


Figure 36. Query service

A query service facilitates reuse of visualization by systemizing querying/answering visual designs that meet the user goals. Answered visual design(s) is used as a key identifier for searching visualization component(s) that are already developed. In addition, a query service is used for evaluating performance visualization ontology. In practical, deciding how well knowledge is designed is up to how well (i.e., correct) knowledge answers queries. A query service enables delivering queries/answers on validating the PVO.

A RMC directory service maintains component information for retrieval by others. For instance, a RMC developer develops a new network monitoring component. This monitoring component is a web service that measures the network traffic of the server on which the component runs. Using a registration web page, the RMC developer submits component information, such as analysis types and event types, with the location of Web Service Description Language(WSDL) [123] that describes how the component interacts. When a RMC directory service finds this RMC best-matched, it returns the location of its WSDL so that a requesting agent accesses the actual network monitoring component based on the WSDL.

In Chapter 5, We have identified different types of composition, ranging from simple to complex: placing two views in the same window and providing view synchronization; merging two or more separate data streams of the same type into a single view; merging two or more data streams of different types into a view connected with one of the streams; and deriving a new view that connects to two or more data streams. A composition service systemizes a construction process of a system-level performance view. It performs *merge or synthesis composition operation*.

7.2 Case Study

This section describes a case study of performance monitoring that demonstrates the research outcomes. The case study is based on the performance scenario introduced in Chapter 1. In this case study, we have developed a simulated environment to obtain necessary performance data.

7.2.1 Performance Scenario

In the performance scenario discussed in Chapter 1, scientists are collaborating to decide the position of solar panel of a space station, running a collaborative VR application on a SGI Onyx at ISU and a Linux cluster at NASA. Both systems are connected by a high-speed vBNS network. As an ISU scientist moves a solar panel virtual object, a NASA scientist experiences an unexpected latency during rendering the panel object. Suppose the following multiple causes across multiple levels and platforms result in the latency:

1. Suppose a certain collaboration application running on SGI Onyx at ISU has been involving a heavy message transmission with the Linux cluster at UCSD. A TCP window size between ISU and UCSD, however, is incorrectly set, which causes TCP packet retransmission.
2. As these retransmitted packets hold message buffers at ISU longer than expected, the memory allocation of these buffers clashes with the memory allocated for other operations.
3. In turn, CPU at ISU dedicates all CPU power to a memory manager to resolve this clash. As a scientist at ISU moves a solar panel object, an object middleware issues an object movement request, which then needs to be translated to TCP/IP transmission to NASA. Meanwhile, the clash delays handling the requests from VR application at ISU.
4. Consequently, the delay results in a high latency in message transmission to the NASA.

When monitoring these events, typical performance visualization environments suffer three problems: (1) they provide little guidance on selection of visualizations; a system manager should have in-depth design knowledge; (2) they are not reusable. A line chart visualization used for monitoring TCP transmission at ISU has little chance to be reused at NASA to monitor incoming traffic; and (3) they support only fixed system-level views. In the given performance scenario, as a root problem is being traced, several system-level views should be created dynamically; e.g., a view showing correlation between process

scheduling and memory utilization, a view showing correlation between memory utilization and network utilization. Existing performance visualizations, however, are limited to fixed system-level views.

URV addresses these problems. Suppose a visualization expert at somewhere already invented a new visualization for CPU load, following URV interface definitions. He/she provides a description of its visualization knowledge via PVO. A system manager in a different domain has performance data whose constraints fit the form of the said description; he/she can apply the same visualization to visualize the CPU load in the new platform.

7.2.2 Testbed

This section describes the testbed that simulates the performance problems mentioned in the previous section. For testbed, we have developed a simulation environment using a simulation package, called Simjava, which is a process based discrete event simulation package [57]. Simjava allows defining simulated entities that sends or/and receives discrete events, and simulated designated operations. A basic operation of simulated entities is communication. Each entity can exchange messages, which describe status/actions of entity. Figure 37 shows the testbed, including simulated entities and a sequence of performance problems.

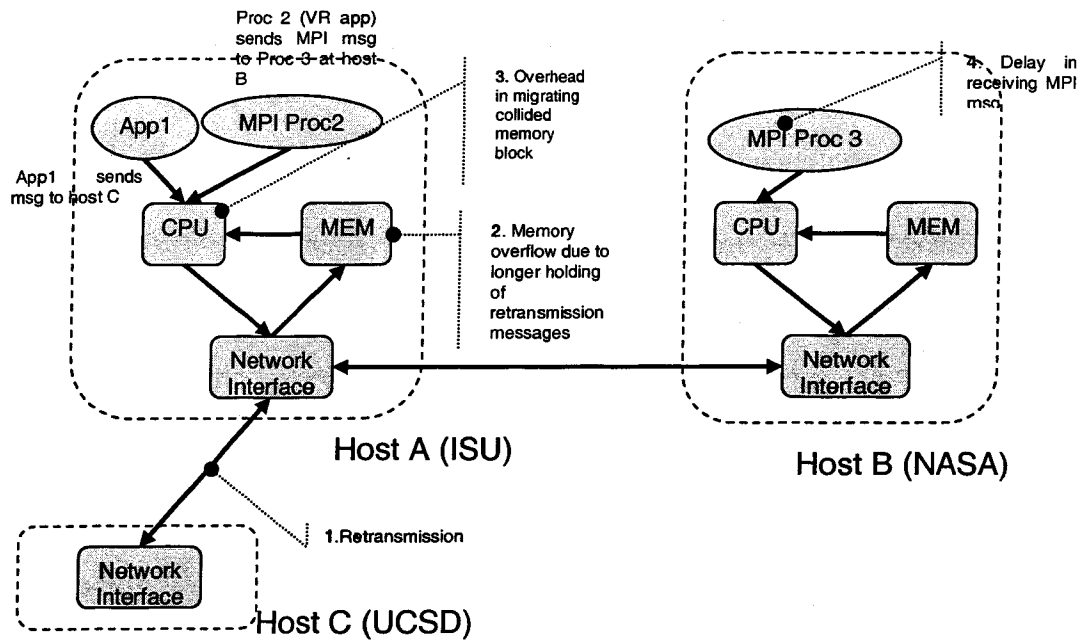


Figure 37. Performance scenario testbed

For instance, a message ID 0 presents a send message from the application 1 in the Figure 37. In addition, Simjava allows setting a probability in a certain even occurrence. We can specify, for instance, a probability of retransmission occurrence.

7.2.3 Problem Diagnosis Scenario

The following sub research outcomes are demonstrated in this section. A full list of research outcomes is described in Chapter 9.

- **Ability to retrieve resource monitoring components of interest**
 - Related contribution: Visualization framework (C.1 in Section 1.5)
- **Ability to identify proper visualization components based on the knowledge**

- Related contribution: Knowledge representation (A.1 in Section 1.5)
- **Ability to compose visualization components dynamically**
 - Related contribution: Composition methodology (B.2 in Section 1.5)
- **Ability to compose visualization components based on their graphical primitives**
 - Related contribution: Visualization component creation methodology (B.1 in Section 1.5)
- **Ability to compose visualization components based on their knowledge**
 - Related contribution: Composition methodology (B.2 in Section 1.5)
- **Ability to manipulate visualization component using SSPV**
 - Related contribution: Spreadsheet visualization (C.2 in Section 1.5)

Using the testbed, we have obtained performance data of each simulated entity, such as CPU load at Host A, memory allocation at Host A, and receive events at Host B. In this section, given performance data, we describe how to drill down a performance problem. Note that there could be many possible ways to drill down the problem. In this research, we do not attempt to provide an optimal drilldown. Instead, we focus on providing a systematic way of drilldown.

Observing the first symptom, which is a latency at Host B (NASA), we can narrow down the issue with following diagnosis steps:

Step1: Observe the status of processes at each host.

First, we can observe the status of process at Host A and B, which involves in object drawing, which sends messages to B. Using the RMC directory browser of SSPV, we first need to identify RMC that can provide such status information (Figure 38).

Name	Target node	Level	Resource	Metric	Service	Port	Method	Location	Select
CPU_B_Load	B-129.196.232.9	OSlevel	CPU	Load	DemoRMC	DemoRMCPRC	get_cpu_b_load	http://localhost:8080/D...	<input type="checkbox"/>
CPU_A_Load	A-129.196.232.9	OSlevel	CPU	Load	DemoRMC	DemoRMCPRC	get_cpu_a_load	http://localhost:8080/D...	<input type="checkbox"/>
P2_Run_Status	A-129.196.232.9	Processlevel	P2	Status	DemoRMC	DemoRMCPRC	get_p2_run_st...	http://localhost:8080/D...	<input checked="" type="checkbox"/>
P3_Run_Status	B-129.196.232.9	Processlevel	P3	Status	DemoRMC	DemoRMCPRC	get_p3_run_st...	http://localhost:8080/D...	<input type="checkbox"/>
P2_Recv_Event	B-129.196.232.9	Processlevel	P2	Recv_Event	DemoRMC	DemoRMCPRC	get_p2_recv_e...	http://localhost:8080/D...	<input type="checkbox"/>
P3_Recv_Event	B-129.196.232.9	Processlevel	P3	Recv_Event	DemoRMC	DemoRMCPRC	get_p3_recv_e...	http://localhost:8080/D...	<input type="checkbox"/>
P2_Send_Event	A-129.196.232.9	Processlevel	P2	Send_Event	DemoRMC	DemoRMCPRC	get_p2_send...	http://localhost:8080/D...	<input type="checkbox"/>
A_Memory_Load	A-129.196.232.9	OSLevel	Memory	Load	DemoRMC	DemoRMCPRC	get_memory_L...	http://localhost:8080/D...	<input type="checkbox"/>
Retrans_AC	AC-129.196.232.9	Networklevel	Link	Retransmission	DemoRMC	DemoRMCPRC	get_retrans_AC	http://localhost:8080/D...	<input type="checkbox"/>
Retrans_AB	AB-129.196.232.9	Networklevel	Link	Retransmission	DemoRMC	DemoRMCPRC	get_retrans_AB	http://localhost:8080/D...	<input type="checkbox"/>

Figure 38. Select a RMC that provides the process status information

Based on the performance data type, SSPV queries PVO for the matched visualization components. Given the process status information, the query service returns a gantt chart component, which can be previewed before added to the spreadsheet (Figure 39). A yellow color (or a lighter shade on the mono color printout of this thesis) of rectangle presents idle while green color (or a darker shade) presents a busy status. The gantt VC is added to SSPV (Figure 40).

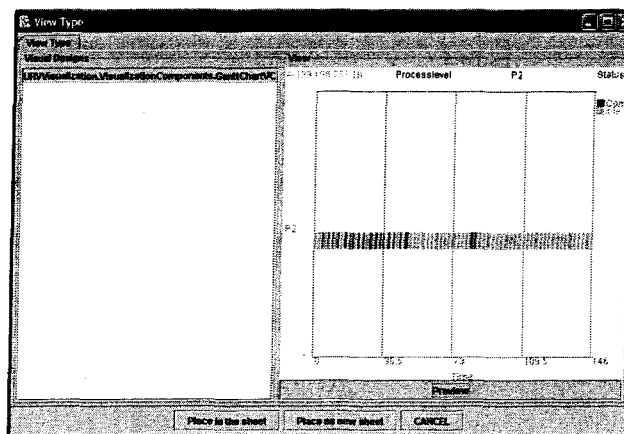


Figure 39. Matched Gantt Chart component for process status

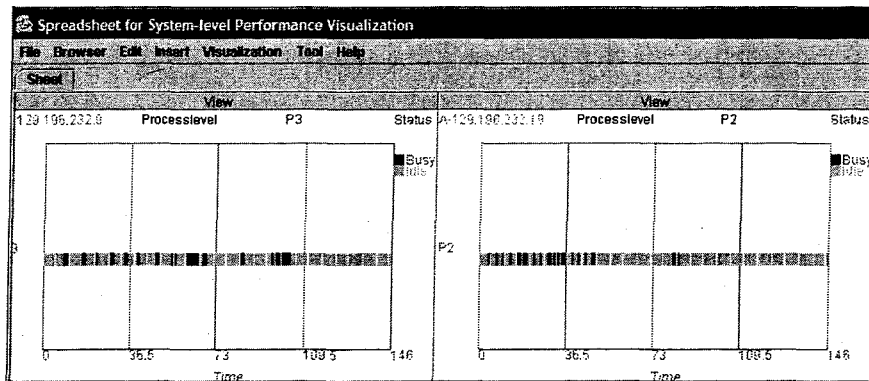


Figure 40. Gantt Chart components added to the worksheet

Research Outcomes Demonstrated in Step 1

- **Ability to retrieve resource monitoring components of interest:** It is able to browse available RMCs that produce process status information.
- **Ability to identify proper visualization components based on the knowledge:** It is able to identify a Gantt chart based on the event type of performance data
- **Ability to manipulate visualization component using SSPV:** It is able to place two different Gantt chart components on the spreadsheet.

Step 2: Attempt to observe the correlation between two process status.

Once we have two separate VCs that monitor process status, we attempt to correlate them into a single view to observe any noticeable behavior. We can merge them into a single view by union.

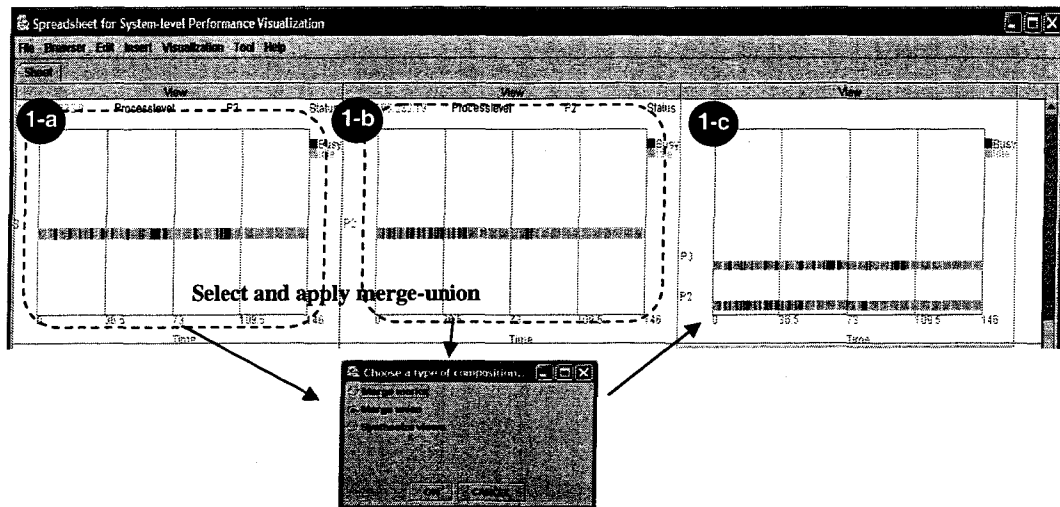


Figure 41. Merge two process status VCs by merge-union

Research Outcomes Demonstrated in Step 2

- **Ability to compose visualization components dynamically:** It is able to create a new visualization on the fly.
- **Ability to compose visualization components based on their graphical primitives:** It is able to compose two Gantt charts by union-merging.
- **Ability to manipulate visualization component using SSPV:** It is able to select multiple visualization components on the worksheet and apply a composition operation.

Step 3: Clarify the correlation.

The merged view can give us some insight on their interaction. Meanwhile, the current view does not clearly give how busy/idle status of sender process ($P2$) affects the status of receiver ($P3$). To address this, we attempt to incorporate communication information between processes. We create two additional views, each of which presents message send or receive events (Figure 42).

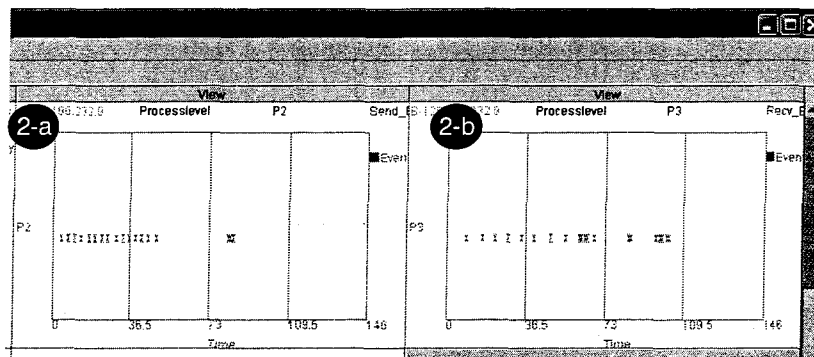


Figure 42. Visualization of Send/Receive events

Then, we incorporate those send/receive event information into the **1-c** view in Figure 41. This can be done by synthesis operation. The knowledge of each VC (**1-c**, **2-a**, **2-b**) is represented as:

- Knowledge of **1-c** = ({Status}, {Process}, {Busy, Idle})
- Knowledge of **2-a** = ({Status}, {Process}, {Send})
- Knowledge of **2-b** = ({Status}, {Process}, {Receive})

A new knowledge can be obtained by synthesizing them: ({Status}, {Process}, {Busy, Idle, Send, Receive}). Using the new composed knowledge, SSPV queries PVO. The matched VC (space-diagram) is created (Figure 43). From the new view, we can observe that the sender (*P2*) is unnecessarily idle between time 50 and 75. This implies that CPU at Host A might be busy doing something else.

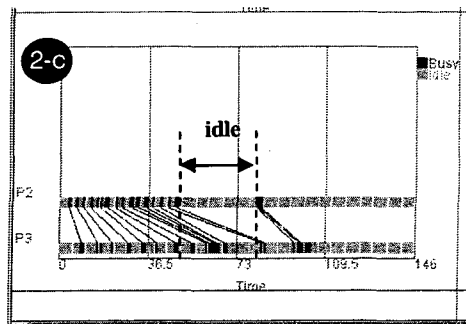


Figure 43. Space diagram visualization component

Research Outcomes Demonstrated in Step 3:

- **Ability to retrieve resource monitoring components of interest:** It is able to browse process send/receive event information via the RMC directory browser.
- **Ability to identify proper visualization components based on the knowledge:** It is able to identify event chart view based on send/receive event data.
- **Ability to compose visualization components dynamically:** It is able to identify a new composite view on the fly.
- **Ability to compose visualization components based on their knowledge:** It is able to identify a new space diagram view based on combined knowledge

Step 4: Correlate CPU information

In order to verify our hypothesis (i.e., CPU at Host A is being hold by something else) in the previous step, we attempt to incorporate CPU load information. First, we create two VCs that present CPU load at each host (Figure 44). Then, we merge those CPU VCs with 2-c in Figure 43 by marks. During the merging-marks, each rectangle, which presents process status, in 2-c, are encoded with additional CPU load value (Figure 44). From the newly composed view, we can confirm that CPU at host A is busy doing

something else, which leads to the conclusion that the root cause of the problem is not in Host B. Instead, the issue resides in Host A.

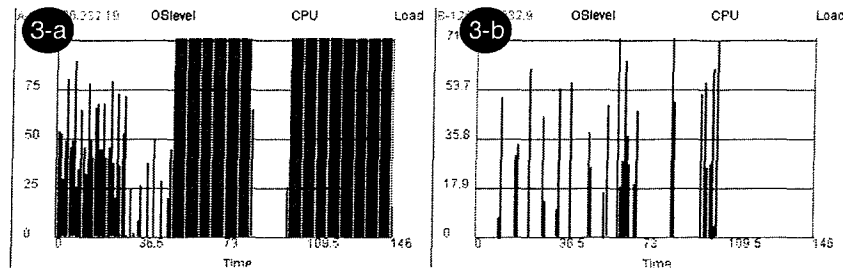


Figure 44. Visualization of CPU load information

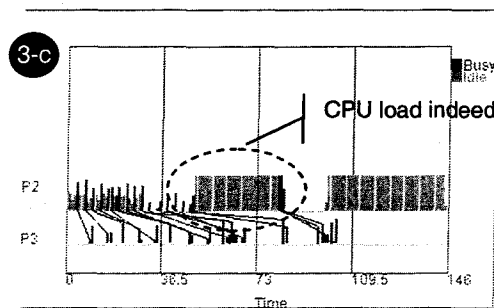


Figure 45. Composed space-diagram with CPU load information

Research Outcomes Demonstrated in Step 4:

- **Ability to retrieve resource monitoring components of interest:** It is able to obtain CPU load information via the RMC directory browser.
- **Ability to identify proper visualization components based on the knowledge:** It is able to identify line chart view for a given CPU load data
- **Ability to compose visualization components based on their graphical primitives:** It is able to compose views by re-encoding process status information with CPU load data.

Step 5: Find a resource that causes CPU load

In this step, we further correlate behavior of resource at Host A. In a real environment, there could be many available RMCs. In this particular testbed, we, however, focus on a limited set of RMCs to evaluate the proposed approach. Among the available RMCs that monitor resources at Host A, we attempt to observe the information about memory allocation for retransmission. 4-a in Figure 46 shows the memory allocation information at Host A. We can align two VCs vertically so that we can observe implicit correlation over the time. As shown in Figure 46, aligning 3-c and 4-a can give us a good visual correlation; CPU load information in 3-c are well matched with the growth of memory allocation. This leads us to the observation that memory buffer allocation due to the retransmission eventually affects the rendering latency in host B.

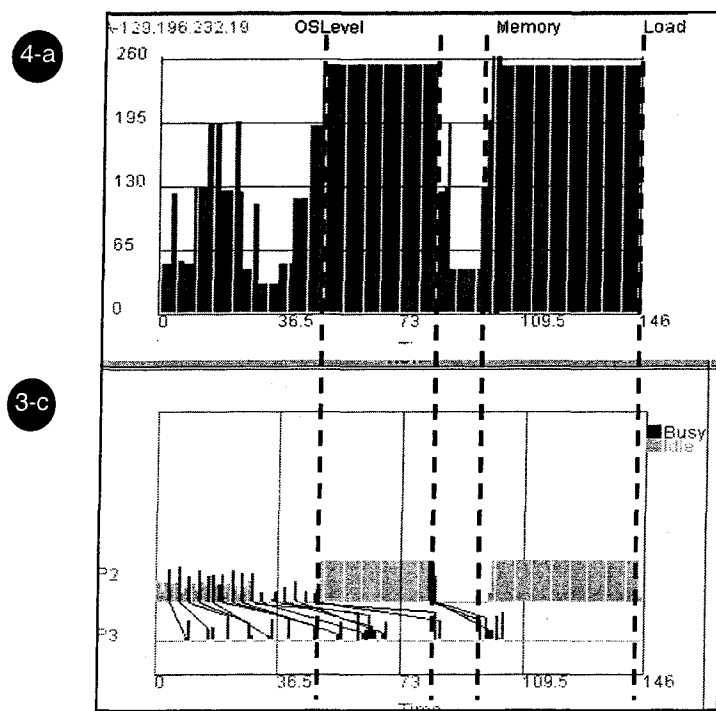


Figure 46. Observing correlation by aligning two VCs vertically on SSPV

Research Outcomes Demonstrated in Step 5:

- **Ability to manipulate visualization component using SSPV:** It is able to align multiple visualization components

Through the problem drilldown processes, we have observed that the URV framework provides robust visualization creation and composition, which help creation of useful system-level visualization effectively. Using the RMC directory service, a user is able to locate resources of interest and access their performance data. Given the type of performance data, URV is able to identify proper visualization component(s) by querying the performance visualization ontology. Such identified visualization components are easily specialized with the actual data and plugged into the SSPV. SSPV provides a greater flexibility in manipulation of multiple visualization components. In particular, we can apply composition operations against multiple visualization components with a few button clicks.

7.3 Summary

This Chapter provided the proposed methodologies, principles, and software with an end-to-end performance monitoring testbed. First, we presented a performance visualization system that is built based on URV framework. The system consists of multiple web services that implement knowledge querying and composition, and SSPV as an end-user monitoring tool. For the testbed, we implemented the performance monitoring scenario introduced in Chapter 1 with a simulation package. The section 7.2.3 described problem drilldown processes in detail, focusing on demonstrating associated research outcomes. Through the case study, this Chapter shows the abilities of retrieving

data of interest, identifying proper visualizations systematically, composing multiple visualizations dynamically based on graphical primitives or knowledge, and manipulating visualizations with the SSPV.

CHAPTER 8. RELATED WORK

This Chapter provides the related work in three relevant research areas. First, we present three different approaches, such as taxonomy, automatic visualization, and component-based visualization, in the area of information visualization that influenced this research. We then review existing performance visualizations. Finally, current performance monitoring issues associated with this research will be discussed with several grid monitoring frameworks and their underlying architectures.

8.1 Visualization Taxonomy

The taxonomy of visualizations helps designers to determine how to visually display information. In the area of information visualization, numerous taxonomies have been developed in terms of a look, a usage, and operators [21][22][24][79][105]. Shneiderman divided information visualization techniques using seven visual data types: temporal, 1D, 2D, 3D, multi-D, tree, and network [105]. He also categorized visualizations by task (e.g., overview, zoom, filter, and so on). As another data-centric taxonomy, Mackinlay and Card described and analyzed portions of design space in order to understand the differences among visualization designs [21]. Using three different data types (nominal, ordered, quantitative) and visual encoding (e.g., color, size), they described the major types of visualizations such as scientific visualization and multi-dimensional tables. Their taxonomies have provided many visualization researches with a data-based visualization selection. This research takes advantages of both Shneiderman's and Mackinlay's approaches in designing a representation of visualization knowledge. Focusing on

performance visualization, Reed and Ribler constructed a taxonomy based on data types [84]. To describe performance views, they defined three metrics: univariate or multivariate, ordinal or categorical, and static or dynamic.

Our research contribution to the visualization taxonomy area is to provide a way of extending the existing visualization categorization to incorporate knowledge. The knowledge of interest as a metric is an applicable problem domain that helps evaluate the usefulness of a visualization. This research further categorizes existing visualizations by their knowledge so that it provides more comprehensive and efficient assistance in visualization selection. In particular, we use a RDF-based representation, which can be shared and manipulated systematically. Such a systematically sharable representation has not been supported before this research. In addition, this research employs a query-answer reasoning process for querying knowledge. By reasoning the query, the system can search matched visualizations more effectively.

8.2 Automatic Visualization

There are two different approaches in automatic visualization: rule-based and example-based. Automatic Presentation Tool (APT) [70] constructs a visual presentation by using a set of predefined design rules that map the data to be conveyed onto desired visual formalisms. In this approach, a visualization design including the perspective and the graphical representations can be codified as sentences in a graphical language. A given set of data is broken down until it matches a primitive language. By composing each primitive language, a final language is constructed; a final visual presentation is created. Exploiting the composition approach of APT, this research extends the APT approach to support knowledge-level composition. Our composition process is similar to

the APT approach in a way that first constructs a representation of composite view. When composing two visualizations, we first merge the knowledge of each visualization. In contrast to APT which always creates a new visualization, our approach finds proper visualizations, which support a new combined knowledge, from the existing visualizations. In such a way, this research has improved the reusability of visualization significantly.

Sage [47] is an example-based visualization system that automatically presents quantitative and relational data using numerous variations and syntheses of 2D static displays. It starts with a set of existing visual presentations (examples). Upon a user's request (e.g., finding presentations that are suitable for my data), the system can search through its example database, and retrieve the most relevant examples. The retrieved examples can then be reused or adapted to the new situation. Sage leverages APT in terms of the number and complexity of graphical displays and data types to be considered. This research then leverages the Sage's approach in a way that supports the knowledge-level reuse of visualization. Both APT and Sage are limited to the data type based visualization creation, which is simple and robust.

Their approaches, however, are not enough when exploited for visualization composition. Since they only consider data type, the semantics of existing visualizations (i.e., what the visualization is about) are lost during composition. Visualization composition in their approaches simply means merging data sets and creating a new visualization, which does not carry over existing knowledge about visualization. For instance, visual elements, which are defined in previous views, are lost in the new composed view; the semantic of visual elements (e.g., a red rectangle means a high CPU load) can be presented with different visual elements (e.g., after composed, the red rectangle CPU load information can be encoded into a network link, which is an incorrect

encoding). This is because that their approaches do not consider the semantic of visual elements and visualizations. As long as the data type is matched, they just encode data into any visual element without semantic validation. In contrast, the new visualization, which is identified via our composition process, reflect all the knowledge of source visualizations. In our approach, before we builds a new composed visualization, we first validates the composition semantically. In other words, we first verifies whether there is proper visualization(s) that can accommodate all composed knowledge, and once verified, we construct the composed view by plugging the visual elements of source views into the newly identified visualization. In that way, the semantics of visual elements and visualizations are preserved in the new composed visualization.

8.3 Component-based Visualization

A component-based framework has been widely used to reuse specific visualization modules. AVS [2] and VTK [56] have been developed based on a pipelined and component-based architecture. These systems provide visualization developers with various software modules, such as data load modules, filtering modules, and rendering modules, allowing the developers quickly to compose them into a final visualization. The systems are flexible in the sense that components can be combined in multiple ways, thereby allowing a wide variety of visualization tasks. In addition, they provide programming interfaces for module developers to add new components to the systems. This feature makes the systems extensible. This research is differentiated from these systems in the sense that it provides sharable visualization knowledge. Neither AVS nor VTK supports a certain methodology for sharing visualization knowledge. Visualization

users have little chance to learn about design principles and knowledge behind the visualizations. Our research addresses this reusability issue.

Snap-Together visualization [77] is a component-based framework that enables users to construct custom exploration interfaces by coordinating multiple views. Using Snap API, visualization developers can make their visualizations snap-able. Once snap-able, the visualizations are controlled by custom exploration methods. This research is differentiated from Snap-Together, in terms of a data domain and guidance availability. Our research focuses on heterogeneous performance data while Snap-Together focuses primarily on a fixed relational data. In addition, in Snap-Together visualization, a visualization developer is in charge of selecting visualization components to be coordinated; no guidance is available. On the other hand, this research provides a systematic guidance on creating visualizations. By reasoning against the performance visualization ontology, our visualization framework suggests users proper visualizations that meet user goals.

The Snap-Together visualization focuses on correlation in visualization control; a user can control multiple visualizations in a single interface. The data presented in each visualization are correlated. In contrast, this research focuses on correlation in visualization itself. We present visual correlation in a single composed view. Regarding the control, this research provides a spreadsheet-based visualization tool that enables manipulation of multiple visualization components. The spreadsheet supports time-based control, which can correlate multiple data at the same timestamp.

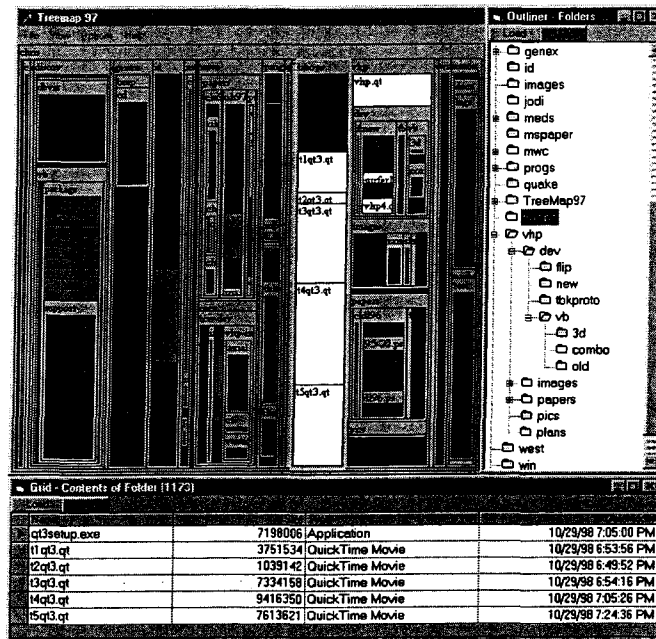


Figure 47. Snapshot of Snap-Together [77]: Snaps multiple visual interfaces (e.g., tree view, table view) to explore file hierarchies.

8.4 Performance Visualization

Rivet [17] is a visualization system for the study of complex computer systems. The goal of Rivet is to provide a rapid development of interactive visualizations for large data sets and to allow users to add new visualization components. Like APT, Rivet breaks down a visualization into graphical primitives and by matching each primitive with data, constructs a new visualization. It uses a component-based architecture and provides a unified platform for the analysis and visualization of computer systems. In addition, like Snap-Together, Rivet provides coordination mechanisms, which add extensive interactivity. Both Rivet and this research employ a modular design in a way that decouples visualization and data collection (e.g., instrumentation). This modular design makes it easy to apply visualizations to wide range of problems. On the other hand, Rivet

does not support dynamic composition; once a visualization is created, it remains a fixed view.

NetLogger Visualization (NLV) [76] is a visualization tool for NetLogger [108]. NetLogger is a performance analysis tool performing detailed end-to-end analysis of distributed applications. It includes instrumentation tools for applications, host systems, and networks. NetLogger is useful for debugging and tuning distributed applications, and detecting bottleneck. The key feature of NLV is to correlate performance data from all own instrumentation. All performance data from different resources are synchronized and viewed in a single window (i.e., a system-level view). The system-level view of NLV, however, is constant; it does not support dynamic integration with newly available performance data. Exploiting synchronization methodology of NLV, this research leverages the current data correlation feature of NLV by creating system-level views dynamically.

Vampir [117] is a performance analysis tool that provides a convenient way to visually analyze runtime event traces produced by MPI applications. Vampir supports various performance visualizations, including a timeline, communication statistics, and execution statistics visualizations. In addition, each visualization supports various user interactions, such as zoom, pan, select, and expand. Vampir, however, is resource- and platform-dependent; it is neither applicable for new problem domains nor operable on different platforms. A fundamental difference is that this research focuses on the uniformity and reusability of visualizations while Vampir focuses on the interactivity and variety of visualizations (i.e., powerful drilldown and more display options).

8.5 Performance Monitoring Tools

In grid environments, there are potentially thousands of distributed resources to be monitored and thousands of entities that use these performance information. A grid monitoring system is differentiated from general monitoring systems in the sense that it must scale across networks as well as accommodate a large number of heterogeneous resources. In the area of grid resource monitoring, several monitoring frameworks have been developed to provide an extensible infrastructure for easy access of distributed performance data.

Grid Monitoring Architecture (GMA) [109] is a new performance monitoring architecture in grid computing that has been proposed by a Global Grid Forum (GGF) performance group. The basic model of GMA consists of three elements: a directory service, a producer, and a consumer. Producers publish their existence in directory service entries. A directory service is used to locate producers and consumers. Consumers can use the directory service to discover the producers of interest. A producer provides performance data and a consumer uses them. GMA influenced a visualization model of this research, which was described in Chapter 2.

Network Weather Service (NWS) [124] is an example of a monitoring framework to support monitoring environments with data coming from measurement sensors. NWS consists of four processes: a persistent state process, a name server process, a sensor process, and a forecaster process. Each distributed sensor process gathers and sends performance measurement (e.g., CPU load, network load) to a persistent state process. A persistent state process stores and retrieves the performance measurement. A name server provides a directory service to bind process and data names with contact information. In addition, using numerical models, a forecaster process produces predictive data. To generate a forecast, the forecaster process uses the relevant measurement history from a persistent state process. The time-based ordered measurements may then be treated as a

time series for forecasting. The main goal of NWS is to provide accurate forecasts of dynamically changing performance characteristics from a distributed set of resources. Both NWS and this research address an information integration issue. Each work, however, has a different purpose; this research focuses on better presentation of a performance problem while NWS focuses on better prediction of the problem.

GridMapper is a tool for monitoring and visualizing the behavior of grid systems [4]. It builds on basic mechanisms for accessing performance information sources and for mapping from domain names to physical locations. The visualization system supports the layout of distributed sets of sources and animation of their behaviors (mostly status). GridMapper focuses on visualizations for the overview of resource behaviors while this research focuses on visualization composition for performance problem drill-downs.

The Information and Monitoring Services Architecture (IMSA) is one of work packages initiated by the DataGrid project [32] [125]. IMSA is a reference implementation of GMA. IMSA uses a relational model to represent performance events and HTTP servlet technology for interoperable communication. This approach has the benefit of being flexible in the query language and simple to build a distributed system. IMSA permits both job performance optimization as well as tracing. The purpose of this work package is to provide a flexible infrastructure to provide easy access to current and archived information about grid resources. This research is differentiated from IMSA in terms of a way of integrating performance data. In IMSA, integration is done at the data level; using a SQL query, system managers integrate multiple performance data of interest in DB. IMSA, however, does not support visualizations for these integrated data. It still relies on resource-specific visualizations with fixed views. In this research, integration is done at the visualization level so that integrated data are directly visible.

These tools and many others provide a wide range of services and capabilities; however, many of them are designed for specific uses and application areas. We use the lessons learned on building and using these tools as the guiding motivation to develop an integrated suite of framework to aid the creation and management of performance visualizations.

CHAPTER 9. CONCLUSIONS

Most performance visualization techniques have focused on specific types of performance events at a certain level of the system and have been invented in isolation without consideration for the interactions and dependencies within the larger system. A distributed application in heterogeneous environments, however, is often engaged with various types of performance problems during its execution across multiple levels of the system. Addressing these, this research presents a new performance visualization framework, called Uniform Resource Visualization (URV), which provides a novel approach for reusable, distributed, and composable visualizations.

9.1 Summary of Contributions

This research contributes to the three areas (qualities of performance visualization, visualization methodology, visualization software) with specific research outcomes.

A. Qualities of a performance visualization

1. *This research extends current visualization representation techniques to the field of performance visualization:* Chapter 2 provides basic information about performance visualization, defining the different aspects of a performance view, including a data, graphical primitive, visual design, and visual element. This definition provides a basis to further define the elements of performance visualization. The elements include a resource, a resource monitoring component, a visualization component, and a monitoring controller.

We formalize the representation of each element so that the knowledge representation about visualization can use. Incorporating existing data-centric visualization categorization, Chapter 3 provides a new knowledge-based visualization categorization. The performance visualization knowledge presents 'what types of resources, performance data type, and/or analysis type have been well presented with a certain visual design'. Chapter 3 presents an ontology-based performance visualization knowledge representation, called Performance Visualization Ontology (PVO). With the PVO, users are able to query well-recognized visual designs for observing a particular performance problem. Users do not need to reinvent a new visual design unless a target problem domain is completely unrelated with ones denoted in the PVO. We present a formal representation of PVO with DAML, which leads to a systematic reuse and manipulation of the knowledge.

B. Visualization methodology

1. *This work creates a methodology for designing reusable visualization components.* Chapter 4 provides the definition of visualization component, including the requirement of building composable and sharable visualization components. In particular, we provide a set of uniform interfaces and data structures of visualization components that are required for information integration into visualization components. Further, we present a visualization component creation methodology, including the guideline of selecting proper graphical primitives. This methodology addresses the heterogeneity in creation of visualization components. A visualization component which is created based on the methodology is able to be consistently accessed and integrated

into the visualization system. The methodology defines several necessary steps: knowledge definition, visual design definition, graphical primitive determination, graphical primitive implementation, template VC creation, and VC specialization. Chapter 4 provides the example of each step. .

2. ***This work provides a methodology for creating system-level views by composition:*** Chapter 5 provides the principle and methodology for system-level view creation. Chapter 5 provides categorization of visualization compositions to help users determine an appropriate composition method to meet the user goals. The levels include a component-level, graphical primitive-level, and a view-level. Further, this research categorizes two composition operations; merge for graphical primitive level and synthesis for view-level compositions. The merge operation is useful for comparing something, and the synthesis is useful for enhancing visual correlation and observing global information. Chapter 5 presents a methodology for synthesis, which is knowledge-based composition. The methodology defines three steps: knowledge composition, visualization search by the composed knowledge, and a final VC creation.

C. Visualization software

1. ***This research presents a spreadsheet-based performance visualization tool:*** Chapter 6 presents a new spreadsheet-based visualization tool, called Spreadsheet for System-level Performance Visualization (SSPV), which provides a robust and flexible visualization environment for dynamic creation of system-level visualizations. It provides a structured way of observing,

exploring, understanding, and managing performance problems. SSPV extends Spreadsheet in Information Visualization (SIV) in a way that supports performance data and composition operations (merge and synthesis). In addition, SSPV supports its own built-in RMC directory browser for easy retrieval of performance data of interest by browsing all available RMCs

2. *This research presents a performance visualization framework:* Chapter 7 presents a performance visualization system. The system is built using URV framework. The framework consists of web services that implement knowledge querying and composition, which provides users with software modules to help create a performance visualization system in a consistent way. To demonstrate the system, we develop a testbed that presents the performance monitoring scenario introduced in Chapter 1. The testbed is built with a simulation package. Chapter 7 also presents the actual problem drilldown processes with the developed visualization system. Through the case study, we show the abilities of retrieving data of interest, systematically identifying proper visualizations, dynamically composing multiple visualizations based on graphical primitives or knowledge, and manipulating visualizations with the SSPV.

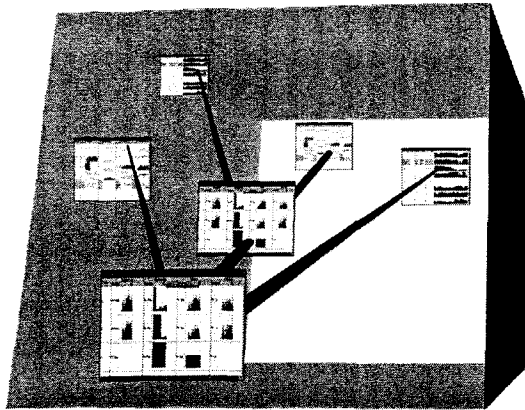
9.2 Future Work

Several extensions of this research include:

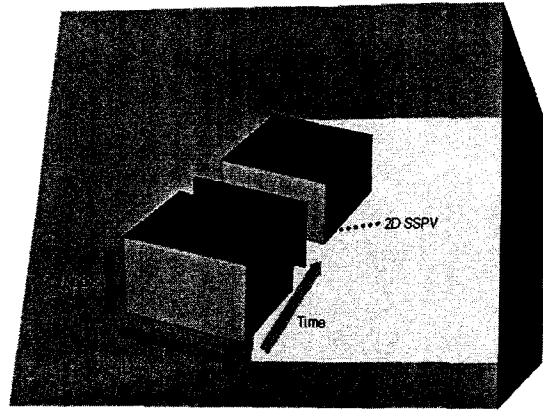
- **Feedback-based Performance Visualization Ontology:** While certain visualization knowledge works well for particular users, it may not well present

some other's performance scenarios. In such cases, we may be able to incorporate user feedback into the PVO. For instance, a user rating value can be added to each visualization instance entry as a weight value which can further narrow down the search.

- **Extension to other visualization domains:** The proposed methodologies and principle are extensible to incorporate other types of visualization. For instance, PVO can be extended to the area of scientific visualizations by identifying key concepts and properties of most scientific visualizations.
- **Extension of Spreadsheet-based System-level Performance Visualization (SSPV):** SSPV can be extended to work in an immersive VR environment, as shown in Figure 48. A VR-based monitoring environment provides additional dimensions for exploring, representing, and interacting with the data. It is particularly effective to maintain a context (or "big picture") while using interactive drilldown to diagnose performance problems. For example in (a), suppose a network manager or chip designer is analyzing the performance of subnets connected with channels. Each SSPV in the C6 presents performance information about the resources in a particular subnet, and the link colors between SSPVs represent channel latencies. As another example in (b), a 3D SSPV is comprised of a set of 2D SSPV slices. Each 2D SSPV presents performance information at a certain time. A user can be immersed in both the time-varying behavior and correlated behavior across nodes and levels.



(a) Linked Multiple SSPVs



(b) 3D SSPV

Figure 48. Extension of SSPV

APPENDIX. PERFORMANCE VISUALIZATION ONTOLOGY

This is the DAML representation of Performance Visualization Ontology that is used in this research.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:ns0="http://www.vrac.iastate.edu/~leekukji/pvo.daml#"
  xmlns:oiled="http://img.cs.man.ac.uk/oil/oiled#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#">
  <daml:Ontology rdf:about="">
    <dc:title>&quot;An Ontology&quot;</dc:title>
    <dc:date></dc:date>
    <dc:creator>Kukjin Lee</dc:creator>
    <dc:description>Performance Visualization
Ontology</dc:description>
  </daml:Ontology>

<!-- Class Definition -->

  <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#BAR_CHART_DESI
GN">
    <rdfs:label>BAR_CHART_DESIGN</rdfs:label>
    <rdfs:subClassOf>
      <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#VISUAL_DESIGN"
/>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#ANALYSIS_TYPE"
>
    <rdfs:label>ANALYSIS_TYPE</rdfs:label>
  </daml:Class>

  <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_DA
TA">
    <rdfs:label>PERFORMANCE_DATA</rdfs:label>
  </daml:Class>

  <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA">
    <rdfs:label>PERFORMANCE_METRIC_DATA</rdfs:label>
    <rdfs:subClassOf>

```

```

        <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_DATA"/>
        </rdfs:subClassOf>
    </daml:Class>

    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#GANTT_CHART_DESIGN">
        <rdfs:label>GANTT_CHART_DESIGN</rdfs:label>
        <rdfs:subClassOf>
            <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#VISUAL_DESIGN"/>
            </rdfs:subClassOf>
        </daml:Class>

<!-- Sub Class Definition -->

    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#VISUAL_DESIGN">
        <rdfs:label>VISUAL_DESIGN</rdfs:label>
        <rdfs:subClassOf>
            <daml:Restriction>
                <daml:onProperty
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#supportAnalysisType"/>
                <daml:hasClass>
                    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#ANALYSIS_TYPE"/>
                    </daml:hasClass>
                </daml:Restriction>
            </rdfs:subClassOf>
            <rdfs:subClassOf>
                <daml:Restriction>
                    <daml:onProperty
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#supportPerformanceData"/>
                    <daml:hasClass>
                        <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_METRIC_DATA"/>
                        </daml:hasClass>
                    </daml:Restriction>
                </rdfs:subClassOf>
            <rdfs:subClassOf>
                <daml:Restriction>
                    <daml:onProperty
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#supportResource"/>
                    <daml:hasClass>
                        <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RESOURCE"/>
                        </daml:hasClass>
                    </daml:Restriction>
                </rdfs:subClassOf>
            </daml:Class>

```

```

    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#SCATTER_PLOT_DESIGN">
    <rdfs:label>SCATTER_PLOT_DESIGN</rdfs:label>
    <rdfs:subClassOf>
      <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#VISUAL_DESIGN"
/>
    </rdfs:subClassOf>
    </daml:Class>

    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#MATRIX_VIEW_DESIGN">
    <rdfs:label>MATRIX_VIEW_DESIGN</rdfs:label>
    <rdfs:subClassOf>
      <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#VISUAL_DESIGN"
/>
    </rdfs:subClassOf>
    </daml:Class>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#SPACE_TIME_DIAGRAM_DESIGN">
    <rdfs:label>SPACE_TIME_DIAGRAM_DESIGN</rdfs:label>
    <rdfs:subClassOf>
      <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#VISUAL_DESIGN"
/>
    </rdfs:subClassOf>
    </daml:Class>

    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PIE_CHART_DESIGN">
    <rdfs:label>PIE_CHART_DESIGN</rdfs:label>
    <rdfs:subClassOf>
      <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#VISUAL_DESIGN"
/>
    </rdfs:subClassOf>
    </daml:Class>

    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#LINE_CHART_DESIGN">
    <rdfs:label>LINE_CHART_DESIGN</rdfs:label>
    <rdfs:subClassOf>
      <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#VISUAL_DESIGN"
/>
    </rdfs:subClassOf>
    </daml:Class>

    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_EVENT_DATA">
    <rdfs:label>PERFORMANCE_EVENT_DATA</rdfs:label>
    <rdfs:subClassOf>

```

```

        <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_DA
TA"/>
        </rdfs:subClassOf>
    </daml:Class>

    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#NETWORK_VIEW_D
ESIGN">
        <rdfs:label>NETWORK_VIEW_DESIGN</rdfs:label>
        <rdfs:subClassOf>
            <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#VISUAL_DESIGN"
/>
                </rdfs:subClassOf>
            </daml:Class>

            <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RESOURCE">
                <rdfs:label>RESOURCE</rdfs:label>
            </daml:Class>

<!-- Property Definition -->

        <daml:ObjectProperty
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#supportPerform
anceData">
            <rdfs:label>supportPerformanceData</rdfs:label>
            <rdfs:range>
                <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_DA
TA"/>
                    </rdfs:range>
                </daml:Class>
            </daml:ObjectProperty>

        <daml:ObjectProperty
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#supportAnalysi
sType">
            <rdfs:label>supportAnalysisType</rdfs:label>
            <rdfs:range>
                <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#ANALYSIS_TYPE"
/>
                    </rdfs:range>
                </daml:Class>
            </daml:ObjectProperty>

        <daml:ObjectProperty
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#supportResourc
e">
            <rdfs:label>supportResource</rdfs:label>
            <rdfs:range>
                <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RESOURCE"/>
                    </rdfs:range>
                </daml:Class>
            </daml:ObjectProperty>

<!-- Instance Definition -->
    <rdfs:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Processor">

```

```

    <rdf:type>
      <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RESOURCE"/>
      </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#QueueLength">
      <rdf:type>
        <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
        </rdf:type>
      </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Delay">
      <rdf:type>
        <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
        </rdf:type>
      </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Start">
      <rdf:type>
        <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_EV
ENT_DATA"/>
        </rdf:type>
      </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Trend">
      <rdf:type>
        <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#ANALYSIS_TYPE"
/>
        </rdf:type>
      </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Write">
      <rdf:type>
        <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_EV
ENT_DATA"/>
        </rdf:type>
      </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Available">
      <rdf:type>
        <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
        </rdf:type>
      </rdf:Description>

```

```

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Software">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RESOURCE"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Relationship">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#ANALYSIS_TYPE"
/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Utilization">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Structure">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#ANALYSIS_TYPE"
/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Comparison">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#ANALYSIS_TYPE"
/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Architecture">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RESOURCE"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#NumberOfHops">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
    </rdf:type>
    </rdf:Description>

```

```

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Memory">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RESOURCE"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Acknowledge">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_EV
ENT_DATA"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Contribution">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#ANALYSIS_TYPE"
/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Status">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#ANALYSIS_TYPE"
/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PacketLoss">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Process">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RESOURCE"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Latency">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
    </rdf:type>

```



```

</rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Interaction">
  <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#ANALYSIS_TYPE"
/>
  </rdf:type>
</rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Hierarchy">
  <rdf:type
rdf:resource="http://www.daml.org/2001/03/daml+oil#Thing"/>
  </rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Bandwidth">
  <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
  </rdf:type>
</rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Disk">
  <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RESOURCE"/>
  </rdf:type>
</rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Link">
  <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RESOURCE"/>
  </rdf:type>
</rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Load">
  <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
  </rdf:type>
</rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Loss">
  <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
  </rdf:type>
</rdf:Description>

```

```

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Read">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_EV
ENT_DATA"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Send">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_EV
ENT_DATA"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Size">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Stop">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_EV
ENT_DATA"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Used">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#OS">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RESOURCE"/>
    </rdf:type>
    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Receive">
    <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_EV
ENT_DATA"/>
    </rdf:type>

```

```

    </rdf:Description>

    <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#CPU">
      <rdf:type>
        <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RESOURCE"/>
        </rdf:type>
      </rdf:Description>

      <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#End">
        <rdf:type>
          <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_EV
ENT_DATA"/>
          </rdf:type>
        </rdf:Description>

      <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#RoundTripTime"
>
        <rdf:type>
          <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PERFORMANCE_ME
TRIC_DATA"/>
          </rdf:type>
        </rdf:Description>

<!-- Visual Design Instance Definition -->

      <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#BarChartInstan
ce">
        <rdf:type>
          <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#BAR_CHART_DESI
GN"/>
          </rdf:type>
          <ns0:supportAnalysisType
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Trend"/>
          <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Bandwidth"/
>
          <ns0:supportResource
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Link"/>
        </rdf:Description>

      <rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#LineChartInsta
nce">
        <rdf:type>
          <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#LINE_CHART_DES
IGN"/>
          </rdf:type>
          <ns0:supportResource
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Processor"/
>

```

```

      <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Bandwidth"/
>
      <ns0:supportAnalysisType
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Trend"/>
    </rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PieChartInstan
ce">
  <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#PIE_CHART_DESI
GN"/>
  </rdf:type>
  <ns0:supportResource
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Memory"/>
  <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Available"/
>
  <ns0:supportAnalysisType
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Contributio
n"/>
  </rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#ScatterPlotIns
tance">
  <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#SCATTER_PLOT_D
ESIGN"/>
  </rdf:type>
  <ns0:supportResource
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Processor"/
>
  <ns0:supportResource
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Memory"/>
  <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Load"/>
  <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Available"/
>
  <ns0:supportAnalysisType
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Relationshi
p"/>
  </rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#GanttChartInst
ance">
  <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#GANTT_CHART_DE
SIGN"/>
  </rdf:type>
  <ns0:supportResource
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Process"/>
  <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Busy"/>

```

```

    <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Idle"/>
    <ns0:supportAnalysisType
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Status"/>
  </rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#SpaceTimeDiagramInstance">
  <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#SPACE_TIME_DIAGRAM_DESIGN"/>
  </rdf:type>
  <ns0:supportResource
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Process"/>
  <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Send"/>
  <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Receive"/>
  <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Busy"/>
  <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Idle"/>
  <ns0:supportAnalysisType
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Status"/>
  <ns0:supportAnalysisType
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Interaction"/>
  </rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#MatrixViewInstance">
  <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#MATRIX_VIEW_DESIGN"/>
  </rdf:type>
  <ns0:supportResource
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Link"/>
  <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Bandwidth"/>
  <ns0:supportAnalysisType
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Interaction"/>
  </rdf:Description>

<rdf:Description
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#NetworkViewInstance">
  <rdf:type>
    <daml:Class
rdf:about="http://www.vrac.iastate.edu/~leekukji/pvo.daml#NETWORK_VIEW_DESIGN"/>
  </rdf:type>
  <ns0:supportResource
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Link"/>

```

```
      <ns0:supportResource
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Processor"/
>
      <ns0:supportPerformanceData
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Bandwidth"/
>
      <ns0:supportAnalysisType
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Status"/>
      <ns0:supportAnalysisType
rdf:resource="http://www.vrac.iastate.edu/~leekukji/pvo.daml#Structure"/
>
    </rdf:Description>

</rdf:RDF>
```

BIBLIOGRAPHY

- [1] R. M. Adler, "Emerging standards for component software," *IEEE Computer*, 28(3), March 1995.
- [2] Advanced Visual Systems, <http://www.avs.com>, April 12, 2006.
- [3] M. Aeschlimann, P. Dinda, Loukas Kallivokas, J. Lopez, B. Lowekamp, and D. O'Hallaron, "Preliminary Report on the Design of a Framework for Distributed Visualization," *Parallel and Distributed Processing Techniques and Applications (PDPTA99)*, Las Vegas, NV, August 1999
- [4] W. Allcock, J. Bester, J. Bresnahan, I. Foster, J. Gawor, J. A. Insley, J. M. Link, and M. E. Papka, "GridMapper: A Tool for Visualizing the Behavior of Large-Scale Distributed Systems," *11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, July 2002, pp 179-187.
- [5] 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.
- [6] R. Ayt, et al. "Simple Case Study of a Grid Performance System," White paper, *Global Grid Forum*, 2002.
- [7] M. Baker and G. Smith, "GridRM: A Resource Monitoring System for the Grid," *International Workshop on Grid Computing*, August 2002.
- [8] M. Baker, B. Carpenter, G. Fox, S. Ko, and S. Lim, "mpiJava: An Object-Oriented Java interface to MPI," *International Workshop on Java for Parallel and Distributed Computing, IPPS/SPDP 1999*, San Juan, Puerto Rico, April 1999.
- [9] A. Bakic, "PGRT-TIE Reference Manual," <http://www.egr.msu.edu/Pgtr/PGRT-TIE-REF/book1.htm>, April 12, 2006.

- [10] A. Bakic, "Visual Object Markup Language (VOML) Reference Manual," <http://www.egr.msu.edu/Pgrt/VOML-REF/dtdelem.htm>, April 12, 2006.
- [11] A. Bakic, "Visual Object Markup Language (VOML) Tutorial," Available on-line from <http://www.egr.msu.edu/pgrt/VOML-TUT/>.
- [12] A. Bakic, M. W. Mutka, and D. Rover, "BRISK: A portable and flexible distributed instrumentation system," In *Proceedings of the IEEE 2nd Merged Symposium IPPS/SPDP 1999 13th International Parallel Processing Symposium & 10th Symposium on Parallel and Distributed Processing*, April 12-16 1999, pp. 387-391.
- [13] A. M. Bakic, M. W. Mutka, and D. Rover, "An On-Line Performance Visualization Technology," In *Proceedings of the IEEE Heterogeneous Computing Workshop, in conjunction with the 2nd Merged Symposium IPPS/SPDP 1999 - 13th International Parallel Processing*, April 12-16 1999, pp. 47-59.
- [14] Z. Balaton, P. Kacsuk, N. Podhorszki and F. Vajda, "Comparison of Representative Grid Monitoring Tools," *Report of the Laboratory of Parallel and Distributed Systems*, Computer and Automation Research Institute of the Hungarian Academy of Sciences, 2000.
- [15] C. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu, "XML-Based Information Mediation with MIX," *ACM Conference on Management of Data*, 1999. Available on-line from <http://www.npaci.edu/DICE/mix-system.html>.
- [16] B. Bederson, J. Meyer, and L. Good, "Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java," Technical Report, CS-TR-4137, UMIACS-TR-2000-30, Department of Computer Science, University of Maryland, College Park, May 2000, Available on-line from <http://www.cs.umd.edu/hcil/jazz/learn/papers/index.shtml>.

- [17] R. Bosch, C. Stolte, D. Tang, J. Gerth, M. Rosenblum, and P. Hanrahan, "Rivet: A Flexible Environment for Computer System Visualization," *Computer Graphics* 34(1), February 2000.
- [18] D. Box, D. Ehnebuske, G. Kakivaya, A. layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," *The World Wide Web Consortium* 2000.
- [19] R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko, and M. Yechuri, "A Component Based Services Architecture for Building Distributed Applications," *Ninth IEEE International Symposium on High Performance Distributed Computing*, August 2000.
- [20] W. J. Brown, R. C. Malveau, H. W. McCormick III, and T. J. Mowbray, "AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis," John Wiley & Sons, Inc., 1998.
- [21] S. K. Card and J. Mackinlay, "The Structure of the Information Visualization Design Space," *IEEE Symposium on Information Visualization*, 1997, pp. 92-99.
- [22] S. M. Casner, "A Task Analytic Approach to the Automated Design of Graphic Presentations," *ACM Transactions on Graphics*, 10(2), April 1991, pp. 111-151.
- [23] E. H. Chi and J. T. Riedl, "An Operator Interaction Framework for Visualization Systems," *IEEE Symposium on Information Visualization (InfoVis 98)*, 1998, pp. 63-70.
- [24] E. H. Chi, "A Taxonomy of Visualization Techniques Using the Data State Reference Model," *Proceedings of the IEEE Symposium on Information Visualization 2000*, 2000, pp. 69-75.
- [25] E. H. Chi, J. Riedl, P. Barry, and J. Konstan, "Principles for Information Visualization Spreadsheets," *IEEE Computer Graphics and Applications (Special Issue on Visualization)*, July/August, 1998, pp. 30-38.

- [26] M. C. Chuah and S. F. Roth, "On the semantic of interactive visualizations," *IEEE Proceedings of Information Visualization, San Francisco*, October 1996, pp. 29-36. Available on-line from <http://www.cs.cmu.edu/Groups/sage/subject.html#sage>.
- [27] Common Component Architecture Forum, <http://z.ca.sandia.gov/~cca-forum/>, April 12, 2006.
- [28] CORBA, <http://www.corba.org>, April 12, 2006.
- [29] DAML + OIL Reference Description, <http://www.w3.org/TR/daml+oil-reference>, April 12, 2006.
- [30] DAML Query Language (DQL) Specification, <http://www.daml.org/2003/04/dql/dql>, April 12, 2006.
- [31] DARPA Agent Markup Language (DAML), <http://www.daml.org>, April 12, 2006.
- [32] DataGrid Project, <http://eu-datagrid.web.cern.ch/eu-datagrid/>, April 12, 2006.
- [33] DQL Implementation, <http://ksl.stanford.edu/projects/dql/>, April 12, 2006.
- [34] DQL: A Query Language for Semantic Web, <http://ksl.stanford.edu/projects/dql/>, April 12, 2006.
- [35] Dynamic Java Proxy Classes, <http://java.sun.com/j2se/1.4/docs/guide/reflection/proxy.html>, April 12, 2006.
- [36] A. Ebert, M. Bender, H. Barthel, and A. Divivider, "Tuning a Component-based Visualization System Architecture by Agents," *International Symposium on Smart Graphics*, March 2001.
- [37] S. G. Eick, "A visualization tool for Y2K," *IEEE Computer*, October 1998, pp.63-69.
- [38] S.G. Eick and Thomas A. Ball, "Software visualization in the large," *IEEE Computer*, 29(4), 1996, pp. 33-43.

- [39] T. Eidson, "Grid Programming Environments: A Component-compatible Approach," White paper, Grid Forum Advanced Programming Models Working Group. Available on-line from <http://www.eece.unm.edu/~dbader/grid/WhitePapers/eidson.pdf>.
- [40] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations," *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, 1997, pp. 365-375.
- [41] I. Foster and C. Kesselman (Editors), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1998.
- [42] I. Foster, J. Insley, G. Laszewski, C. Kesselman, and M. Thiebaut, "Distance Visualization: Data Exploration on the Grid," *IEEE Computer*, 32(12), December 1999, pp. 36-43.
- [43] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural Mismatch or, Why it's Hard to Build Systems out Existing Parts," *Proceeding of the 17th Int. Conf. On Software Engineering*, April 1995.
- [44] G. A. Geist, J. A. Kohl, and P. M. Papadopoulos, "CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications," *International Journal of High Performance Computing Applications*, 11(3), August 1997, pp. 224-236.
- [45] Global Grid Forum, <http://www.gridforum.org>, April 12, 2006.
- [46] Globus Heartbeat Monitor Specification, http://www-fp.globus.org/hbm/heartbeat_spec.html, April 12, 2006.
- [47] J. Goldstein, S.F. Roth, J. Kolojechick, and J. Mattis, "A framework for knowledge based interactive data exploration," *Journal of Visual Languages and Computing*, 5, 1994, pp. 339-363.

- [48] D. Gunter and W. Smith, "Schemas for Grid Performance Events," Grid Performance Working Group White Paper, October 2000, Available on-line from <http://www-didc.lbl.gov/GridPerf/papers/EventSchema.pdf>.
- [49] P. Hartling, C. Just, and C. Cruz-Neira, "Distributed Virtual Reality Using Octopus," *Proceedings of the IEEE Virtual Reality 2001 Conference*, March 2001, Yokohama, Japan.
- [50] C. Healey, R. Amant, and M. Elhaddad, "ViA: A perceptual visualization assistant," In *28th Workshop on Advanced Imagery Pattern Recognition (AIPR-99)*, Washington, DC, 1999.
- [51] M. T. Heath and J. E. Finger, "ParaGraph: A tool for visualizing performance of parallel programs," Technical report, Oak Ridge National Laboratory, Oak Ridge, TN, 1994.
- [52] M. T. Heath, A. Malony, and D. Rover, "The Visual Display of Parallel Performance Data," In *IEEE Computer, Special Issue on Performance Evaluation Tools for Parallel and Distributed Computer Systems*, 28(11), November 1995, pp. 21-28.
- [53] M. T. Heath, A. Malony, and D. Rover, "Parallel Performance Visualization: From Practice to Theory," *IEEE Parallel & Distributed Technology*, 3(4), Winter 1995.
- [54] H. H. Hersey, S. T. Hackstadt, L. T. Hansen, and A. Malony, "Viz: A Visualization Programming System," Technical Report CIS-TR-96-05, Department of Computer and Information Science, University of Oregon, Eugene, OR 97403-1202, April 1996.
- [55] W. Hibbard, "Building 3-D User Interface Components Using a Visualization Library," *Computer Graphics*, 36(1), 2002, pp. 4-7.

- [56] D. Hill and M. Wasilewski, "Development of a Component-Based Visualization Environment Using the Visualization Toolkit (VTK)," *Proceedings of the Visualization Development Environments*, 2000.
- [57] F. Howell and R. McNab, "simjava: a discrete event simulation package for Java with applications in computer systems modeling," In *Proceedings of First International Conference on Web-based Modelling and Simulation*, San Diego CA, January 1998.
- [58] W. Hürsch and C. Lopes, "Separation of Concerns," Northeastern University technical report NU-CCS-95-03, Boston, February 1995.
- [59] JavaBeans, <http://java.sun.com/products/javabeans/>, April 12, 2006.
- [60] Jpython, <http://www.jpython.org>, April 12, 2006.
- [61] T. Kamada and S. Kawai, "A General Framework for Visualizing Abstract Objects and Relations," *ACM Transactions on Graphics*, 10(1), January 1991, pp. 1-39.
- [62] K. Karavanic, J. Myllymaki, M. Livny, and B. Miller, "Integrated Visualization of Parallel Program Performance Data," *Special issue on environments and tools for parallel scientific computing, Parallel Computing 23*, 1997.
- [63] S. Kerpedjiev, S. Roth, and J. Mattis, "Functional Unification Approach to Automated Visualization Design," *International Symposium on Smart Graphics*, Mar 2000.
- [64] K. Lee and D. Rover, "A Component-based Framework for Uniform Resource Visualization," *Workshop on Software Visualization, International Conference on Software Engineering (ICSE 2001)*, Toronto, Canada, May 2001.
- [65] K. Lee and D. Rover, "A Component-based Framework for Uniform Resource Visualization," In *Poster Proceedings of IEEE Symposium on Information Visualization (InfoVis 2001)*, San Diego, October 2001.

- [66] K. Lee and D. Rover, "Uniform Resource Visualization (URV): Services and Software," *Dagstuhl Seminar on Performance Analysis and Distributed Computing*, Waden, Germany, August 2002.
- [67] K. Lee and D. Rover, "Uniform Resource Visualization," *Industrial Grid Summit*, Paris, France, June 2001.
- [68] K. Lee and D. Rover, "A Web Services and Ontology Based Performance Visualization Framework for Grid Environments", *IEEE International Conference on Cluster Computing*, Boston, September 2005.
- [69] M. Levoy, "Spreadsheet for images," In *Computer Graphics (SIGGRAPH '94 Proceedings)*, volume 28, SIGGRAPH, 1994, pp. 139 – 146.
- [70] J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Transactions on Graphics*, 5, 1986, pp. 110–141.
- [71] A. Malony, D. Brown, S. Hackstadt, and B.Mohr. Program analysis environments for parallel language systems: The TAU environment. In *Proceedings of the Second Workshop on Environments and Tools for Parallel Scientific Computing*, May 1994, pp. 162-171.
- [72] K. D. Miceli, "A Framework for the Design of Effective Graphics for Scientific Visualization," NASA Ames Research Center, NAS Systems Division, Applied Research Branch technical report RNR-92-035, December 1992.
- [73] Microsoft COM, <http://www.microsfot.com/com>, April 12, 2006.
- [74] B. Miller, J. Cargille, R. Irvin, K. Kunchithapadam, M. Callaghan, J. Hollingsworth, K. Karavanic, and T. Newhall, "The Paradyn parallel performance measurement tools," *IEEE Computer*, 28(11), November 1995, pp. 37-46.
- [75] Z. Nemeth, "Performance Evaluation on Grids: Directions, Issues, and Open Problems," *Report of the Laboratory of Parallel and Distributed Systems*, 2002.

- [76] Netlogger Visualization, <http://www-didc.lbl.gov/NetLogger/nlv/nlvmain.html>, April 12, 2006.
- [77] C. North, A “User Interface for Coordinating Visualizations based on Relational Schemata: Snap-Together Visualization,” University of Maryland Computer Science Dept. Doctoral Dissertation, May 2000. Available on-line from <ftp://ftp.cs.umd.edu/pub/hcil/Reports-Abstracts-Bibliography/abstracts/2000-15.abstract>.
- [78] N. Noy and D. McGuinness, “Ontology Development 101: A Guide to Creating Your First Ontology,” http://protégé.standord.edu/publications/ontology_development/ontology101.html, April 12, 2006.
- [79] OLIVE: On-line. Library of Information Visualization Environment, <http://otal.umd.edu/Olive/>, April 12, 2006.
- [80] Open Lightweight Directory Access Protocol (LDAP), <http://www.openldap.org/>, April 12, 2006.
- [81] Parallel tools consortium, <http://www.ptools.org>, April 12, 2006.
- [82] J. M. Purtillo, “The polyolith software bus,” *ACM Transactions on Programming Languages and Systems*, 16(1), January 1994, pp. 151-174.
- [83] D. Reed, R. Ayt, R. Noe, P. Roth, K. Shields, B. Schwartz, and L. Tavera. “Scalable Performance Analysis: The Pablo Performance Analysis Environment,” In *Proceedings of the Scalable Parallel Libraries Conference. IEEE Computer Society*, 1993.
- [84] D. Reed and R. Ribler, “Performance Analysis and Visualization,” *Computational Grids: State of the Art and Future Directions in High-Performance Distributed Computing*, I. Foster and C. Kesselman (eds), Morgan-Kaufman Publishers, August 1998, pp. 367-393.

- [85] D. Reed, K. Shields, W. Scullin, L. Tavera, and C. Elford, "Virtual reality and parallel systems performance analysis," *IEEE Computer*, 28(11), November 1995, pp. 55-67.
- [86] L. Renambot, T. Schaaf, H. Bal, D. Germans, and H. Spoelder, "Griz: Experience with Remote Visualization over Optical Grid," *iGrid 2002*, September 2002.
- [87] R. Ribler, J. Vetter, H. Simitci, and D. Reed, "Autopilot: Adaptive Control of Distributed Applications", *Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing*, Chicago, IL, July 1998.
- [88] L. Rising, *The Patterns Handbook: Techniques, Strategies, and Applications*, Cambridge University Press, 1998.
- [89] B. Rogowitz and L. Treinish, "An Architecture for Rule-Based Visualization," In *Proceedings of IEEE Visualization '93*, 1993, pp. 236-243.
- [90] S. Roth , M. Chuah , S. Kerpedjiev, J. Kolojejchick, and P. Lucas, "Towards an information visualization workspace: Combining multiple means of expression," *Human-Computer Interaction Journal*, 1997
- [91] S. Roth and J. Mattis, "Data Characterization for Intelligent Graphics Presentation," In *Proceedings SIGCHI'90 Human Factors in Computing Systems*, Seattle, WA, ACM, April 1990, pp. 193-200.
- [92] S. Roth and J. Mattis. "Automating the Presentation of Information," *Proceedings of the IEEE Conference on Artificial Intelligence Applications*, Miami Beach, FL, February 1991, pp. 90-97.
- [93] D. Rover, "Vista: Visualization and Instrumentation of Scalable Multicomputer Applications," *IEEE Parallel and Distributed Technology: Systems and Applications*, 1(3), August 1993, p. 83.

- [94] D. Rover, A. Waheed, M. Mutka, and A. Bakic', "Software Tools for Complex Distributed Systems: Toward Integrated Tool Environments," *IEEE Concurrency*, 6(2), April-June 1998.
- [95] D. Rover, *Encyclopedia of Distributed Computing*, chapter Program Visualization, Kluwer Academic Publishers, 1999.
- [96] D. Rover, Matt W. Mutka, Kurt Stirewalt, and L. Kempel, "Performance visualization: Uniform resource visualization, 2000," Available on-line from <http://www.egr.msu.edu/Pgrr/URV/urvsldes.pdf>.
- [97] H. Rzepa and P. Murray-Rust, "Chemical Rendering Using CML and SVG," <http://www.ch.ic.ac.uk/svg/>, April 12, 2006.
- [98] L. Salisbury, "Automatic Visual Display Design and Creation," Ph.D. thesis, University of Washington, Department of Computer Science and Engineering, 2001.
- [99] Scalable Vector Graphics (SVG), <http://www.w3.org/Graphics/SVG/Overview.htm>, April 12, 2006.
- [100] J. Schneider, "Components, Scripts, and Glue: A conceptual framework for software composition," Ph.D. thesis, University of Bern, Institute of Computer Science and Applied Mathematics, October 1999.
- [101] H. Senay and E. Ignatius, "Rules and principles in scientific data visualization," Technical report, Department of Computer Science, George Washington University, Washington, D.C, 1990.
- [102] H. Senay and E. Ignatius, "VISTA: Visualization Tool Assistant for Viewing Scientific Data," *SIGGRAPH 1990 Course Notes: Data Visualization*, 27(5), 1990, pp. 21-25.

- [103] E. Shaffer, S. Whitmore, B. Schaeffer, and D. Reed, "Virtue: Immersive Performance Visualization of Parallel and Distributed Applications," *IEEE Computer*, December 1999, pp. 44-51.
- [104] J. Shalf and W. Bethel, "How the Grid Will Affect the Architecture of Future Visualization Systems," Visualization Viewpoints column, *IEEE Computer Graphics and Applications*, May/June 2003.
- [105] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualization," *Visual Languages* 96, 1996.
- [106] W. Smith, D. Gunter, and D. Quesnel, "An XML-Based Protocol for Distributed Event Services," In *Proceedings of the 2001 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2001, pp. 1668-1674
- [107] B. Spitznagel and D. Garlan, "A Compositional Approach for Constructing Connectors," *Proceeding of Working IEEE/IFIT Conference On Software Architecture*, 2001.
- [108] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, and M. Thompson, "A Monitoring Sensor Management System for Grid Environments," In *Proceedings of the IEEE High Performance Distributed Computing conference (HPDC-9)*, August 2000.
- [109] B. Tierney, R. Ayt, D Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski, "A Grid Monitoring Architecture," White paper, Global Grid Forum, 2002. <http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf>, April 12, 2006.
- [110] B. Topol, J. Stasko, and V. Sunderam, "PVaniM: A Tool for Visualization in Network Computing Environments," *Concurrency: Practice & Experience*, 10(14), 1998, pp. 1197-1222.
- [111] W. Tracz, "Where does reuse start?," In *Proceedings of Realities of Reuse Workshop*, 1990.

- [112] L. Treinish, "A Function-Based Data Model for Visualization," In *Proceedings of the IEEE Computer Society Visualization 99 Late Breaking Hot Topics*, October 1999, pp. 73-76.
- [113] H. Truong, T. Fahringer, F. Nerieri, and S. Dustdar, "Performance Metrics and Ontology for Describing Performance Data of Grid Workflows," *IEEE International Workshop on Grid Performance co-located at the IEEE Cluster Computing and Grid 2005 Conference (CCGrid)*, May 2005.
- [114] L. Tweedie, "Characterizing Interactive Externalizations," In *Proceedings ACM CHI'97*, 1997, pp.375-382.
- [115] L. Tweedie, "Describing Interactive Visualization Artifacts," in *FADIVA 3*, T. Catarci, Ed. Gubbio, Italy, 1996, pp. 63-66.
- [116] Unified Modeling Language (UML), <http://www.uml.org>, April 12, 2006.
- [117] Vampir, <http://www.pallas.com/e/products/vampir/index.htm>, April 12, 2006.
- [118] A. Varshney and A. Kaufman, "FINESSE: A financial information spreadsheet," *IEEE Information Visualization Symposium*, 1996, pp. 70 – 71.
- [119] Visual Insights (Lucent Technologies). Software components, 1998. <http://www.visualinsights.com/components/>, April 12, 2006.
- [120] VR Juggler, <http://www.vrjuggler.org>, April 12, 2006.
- [121] A. Waheed, W. Smith, J. George, and J. Yan, "An Infrastructure for Monitoring and Management in Computational Grids," In *Proceedings of the Conference on Languages, Compilers, and Runtime Systems*, 2000.
- [122] Web Service Architecture, <http://www.w3.org/TR/ws-arch/>, April 12, 2006.
- [123] Web Service Description Language, <http://www.w3.org/TR/wsdl>, April 12, 2006.

- [124] R. Wolski, N. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Future Generation Computing Systems*, 1999.
- [125] WP3, Information and Monitoring Services, DataGrid Project, <http://hepunix.rl.ac.uk/edg/wp3/>, April 12, 2006.
- [126] O. Zaki, E. Lusk, W. Gropp, and D. Swider. "Toward Scalable Performance Visualization with Jumpshot," *High-Performance Computing Applications*, 13(2), 1999, pp. 277-288.
- [127] P. Zave and M. Jackson, "Conjunction as Composition," *ACM Transactions on Software Engineering and Methodology*, 2(4), October 1993, pp. 379-411.
- [128] P. Zave, "A compositional approach to multi-paradigm programming," *IEEE Software*, 6(5), September 1989.
- [129] J. Zhang, "A representational analysis of relational information displays," *International Journal of Human-Computer Studies*, 44, 1994.
- [130] M. Zhou and S. Ma, "Representing and Retrieving Visual Presentations for Example-Based Graphics Generation," *International Symposium on Smart Graphics*, March 2001.
- [131] M. Zhou and S. Ma, "Representing and retrieving visual presentations for example-based graphics generation," In *Proceedings of Smart Graphics '01*, 2001, pp. 87-94.