4-16-2001

A performance measurement tool for the resource access decider authorization service prototype

Suresh R. Chegireddy
Florida International University

DOI: 10.25148/etd.FI14060158

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

A PERFORMANCE MEASUREMENT TOOL

FOR THE RESOURCE ACCESS DECIDER AUTHORIZATION SERVICE PROTOTYPE

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Suresh R. Chegireddy

2001

To: Dean Arthur W. Herriott
College of Arts and Science

This thesis, written by Suresh R. Chegireddy, and entitled A Performance Measurement Tool for the Resource Access Decider Authorization Service Prototype, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

Prabakar Nagarajan

Shu-Ching Chen

Yi Deng, Major Professor

Date of Defense: April 16, 2001

The thesis of Suresh R. Chegireddy is approved.

Dean Arthur W. Herriott
College of Arts and Sciences

Interim Dean Samuel S. Shapiro
Division of Graduate Studies

Florida International University, 2001

DEDICATION

I dedicate this thesis to my friends, my parents and myself for the help and support without which the completion of this work would not have been possible.

ACKNOWLEDGMENTS

I wish to thank the members of my thesis committee for their guidance, patience and continuos support. Dr. Nagarajan Prabakar has been helpful in providing excellent feedback through different stages of the work. Dr. Shu-Ching Chen has been very cooperative and always available for providing help. Finally, I would like to thank my major professor and advisor, Dr. Yi Deng. He has been the driving force for the completion of this work. His accurate suggestions and expert advice have seen me through the critically important aspects of my thesis.

ABSTRACT OF THE THESIS

A PERFORMANCE MEASUREMENT TOOL

FOR THE RESOURCE ACCESS DECISION AUTHORIZATION SERVICE PROTOTYPE

by

Suresh R. Chegireddy

Florida International University, 2001

Miami, Florida

Professor Yi Deng, Major Professor

The RAD (Resource Access Decider) authorization service is implemented at CADSE (Center for Advanced Distributed Systems Engineering) as a prototype based on the OMG (Object Management Group) CORBA (Common Object Reference Broker Architecture) specification for RAD (Resource Access Decider) facility. It is a part of the research towards developing performance efficient and available distributed authorization service. In order to test the performance of such an implementation, measurements have to be made to obtain data, which can be used to analyze the resource consumption and system behavior under different configurations. Such tests will have to be performed throughout the development process and this requires automating the performance measurement process to streamline the different stages of data gathering, analysis and interpretation. This thesis presents a performance measurement tool based on the DOVE (Distributed Object Visualization Environment) framework, capable of running the tests, gathering the data, computing and interpreting the data into graphical formats. The tool also provides a view of the system behavior by monitoring the performance of the individual components within the RAD prototype.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

1. Introduction

A Performance measurement tool refers to the programs that carry out tests on another application or system with an objective of gathering enough information to measure the performance and behavior of that application or system. A resourceful automated tool usually with a Graphical User Interface (GUI) providing the user with facilities to selectively view the application's performance data such as for e.g. response time, etc., during and after the application execution. The performance measurement tool is a more specific subset of a visualization and monitoring system, which is defined as a "window into an application" by Tuchman et al in [5], thus serving as no more than a viewing utility.

Performance measurement tools usually aid in gathering, computing application data and graphically representing (as graphs and other diagrams) the application's performance for the user to get a better understanding and insight towards developing better /efficient applications.

1.1 Performance Measurement Tools – Purpose and objectives

The purpose of the performance measurement tools is to test the candidate application and gather the resultant data to analyze its performance. A majority of the current applications and software systems are distributed systems. The applications usually run / support small to large networks and performance becomes critically important in such scenarios. The performance measurement tool should be capable to test such a system in terms of issues such as resource consumption and context-based behavior. An example may be to measure the response time of an application under different configurations of its components and services. The tool is required to be automated in order to reduce the time and effort that may be needed in carrying out large jobs by traditional approaches such as command line processing, etc.

The tool should be a well integrated with the complete set of sub systems that are capable of carrying out the different sub tasks of the performance measurement process like data collection, computation and interpretation.

1.2 Problem Description

The Resource Access Decider (RAD) authorization service for Common Object Request Broker Architecture (CORBA) based distributed systems allows decoupling authorization logic for access to resources, from the application's basic functionality [1]. The Center for Advanced Distributed Systems Engineering (CADSE) at Florida International University (FIU) has been actively researching into developing performance efficient authorization systems towards supporting applications in enterprise wide distributed systems with access control capabilities. The research involves development of a prototype of the CORBA RAD facility, which is the Object Management Group (OMG, [9]) specification for a generic authorization framework.

The objective has been to develop a performance efficient and available implementation of the RAD specification as an authorization service. This requires strenuous performance testing of different implementations of the RAD until a reasonably optimum system could be implemented. The performance measurements till date have been performed in a lengthy data collection process on the implementation during request processing and storing the data obtained in one stage. The second stage involved computing the data and interpreting it into graphical plots or performance characteristics to assess the performance and behavior under different configurations. The process requires various applications at different stages and is particularly tedious for the number of times the tests have to be performed in order to obtain consistent and reliable data and the amount of data to be handled during each repetition.

The performance measurement tool presented in this thesis proposes a solution to this problem. The tools integrates the various stages of the process into a single system thus reducing

the overhead of multiple applications, resources and above all the time involved in conducting such measurements on implementations such as the RAD authorization service which are capable of providing the services in a distributed environment. The tool alike the application is also capable of executing in distributed environment to offer its integrated services in performance testing of the RAD prototype. The tool is an implementation of the Distributed Object Visualization Environment (DOVE) framework developed at Washington University. The DOVE framework is a generic architecture for monitoring and measurement systems that can operate and provide their services in distributed environment. The architecture and components of the DOVE framework are explained in detail in Chapter 3.

1.3 RAD Authorization Service and Performance Measurement

Applications are required to provide fine-grained and/or coarse-grained access control to the underlying resources by following certain access control policies. Complex access control policies might require developers to embed the domain-specific authorization logic into application.

The RAD authorization service design described in [1] avoids such complexity to separate the application logic from authorization decision making. A well-designed interface has been specified for this service and submitted to the OMG as a corbamed facility [9]. The specification for the authorization service is claimed to be capable to implement and support any of the contemporary access control mechanisms [11], [12]. The authorization server would provide access control decisions for the application as indicated in Figure 1.

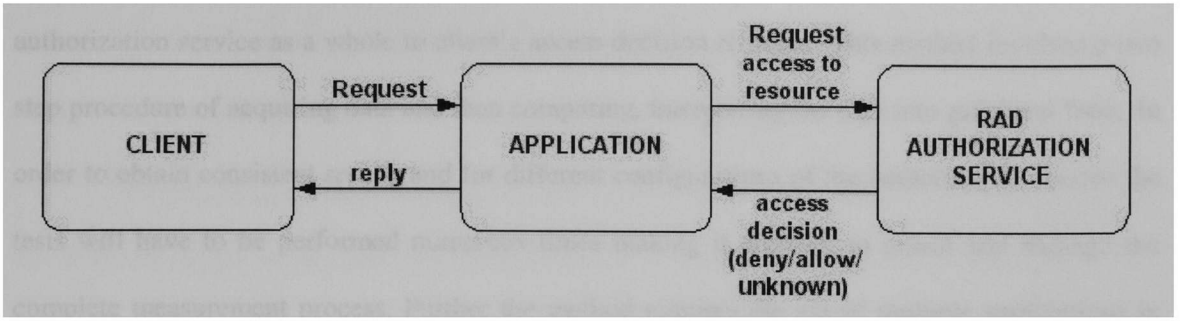

Figure 1. Utility of the RAD Authorization Service

In order to research and develop an efficient authorization service based on the framework, CADSE developed a prototype for the design outlined in [1], as part of its ongoing research in distributed systems. The prototype is a part of CADSE's research in the validity of the of RAD framework in distributed environment access control and the implementation of extensible, maintainable solutions for authorization decision problems, as stated in [10]. The performance and scalability aspects of the service require carrying out performance measurements for the RAD prototype and study the results obtained. In order to explore the scalability and availability aspects of the RAD server prototype a more available implementation of the prototype is being developed and preparations are being made to make performance measurements on this implementation as well.

1.4 Significance of the Work

The development of several implementations of the RAD prototype and the consequent performance measurements is an incremental process for obtaining efficient authorization services. The repeated requirement of performance measurements and study of the different RAD implementations thus calls for an automated tool as more efficient approach to the procedure.

Existing methodology of performance measurement involves executing specially written programs executed at command line to gather data related to the response time of the

authorization service as a whole to client's access decision requests. This method involves a two step procedure of acquiring data and then computing, interpreting the data into graphical form. In order to obtain consistent results and for different configurations of the authorization service the tests will have to be performed numerous times making it difficult to repeat and manage the complete measurement process. Further the method requires the aid of multiple applications in order to compute and interpret the data.

The performance measurement tool makes a significant contribution here by automating the process and reducing the amount of time, effort and resources involved in the complete process. Further, the tool adds capabilities to monitor the internal behavior of the RAD components in the implementation by providing data such as time consumption by individual RAD components.

Properties	**Existing method for performance measurement**	**Performance Measurement Tool/System**
Time consumed / response time	Includes the following stages: data collection, computation, analysis and plotting	Data collection, computation and plotting are achieved by one integrated system.
Data	Data is available for the authorization service as a whole	Data is available for individual components, methods and the authorization service.

Table 1. Matrix of comparison for the Performance Measurement Tool

The performance measurement tool compares to the existing methodology of performance measurement in different aspects, as shown in Table 1. The matrix could be used to

estimate the effectiveness and contribution of the performance measurement tool for the purpose as well as to compare with any other solutions for a similar problem.

The performance measurement tool serves as a single integrated system that can execute the different stages of the performance measurement procedure on RAD authorization service prototype. The system will prove to be a vital research tool for further implementations of RAD in relation to performance measurements and behavior monitoring.

2. Background Information

A majority of current performance measurement tools are a part of larger systems meant for network management and distributed system visualization systems. These systems are designed to extract resource consumption data form the applications being monitored and tested in distributed environments.

This chapter provides an overview of the paradigm of such systems. Some background information about monitoring and visualization systems in general and a few well-developed systems are presented in the next two sections. The last section lists the desirable properties of performance measurement tool in general as well as with respect to the RAD implementation.

2.1 Performance Measurement Paradigm and Existing Tools

This section presents the basic process flow architecture of performance measurement tools in general in the form of the data flow paradigm.

2.1.1 Dataflow Paradigm

The dataflow paradigm represents the process of data collection from applications and the subsequent data processing and interpretation as a process flow model shown in Figure 3. The data source is the application while the processing modules indicate the internal computations and analyses carried out by the performance tool internally before representing the data in the graphical forms such as xy-plots, etc. Thus the whole system would be composed of sub units or modules, that can be customized to fit into the network of modules that a generic performance measurement system would be composed of.

The dataflow paradigm is represented by the process flow diagram shown in Figure 2.

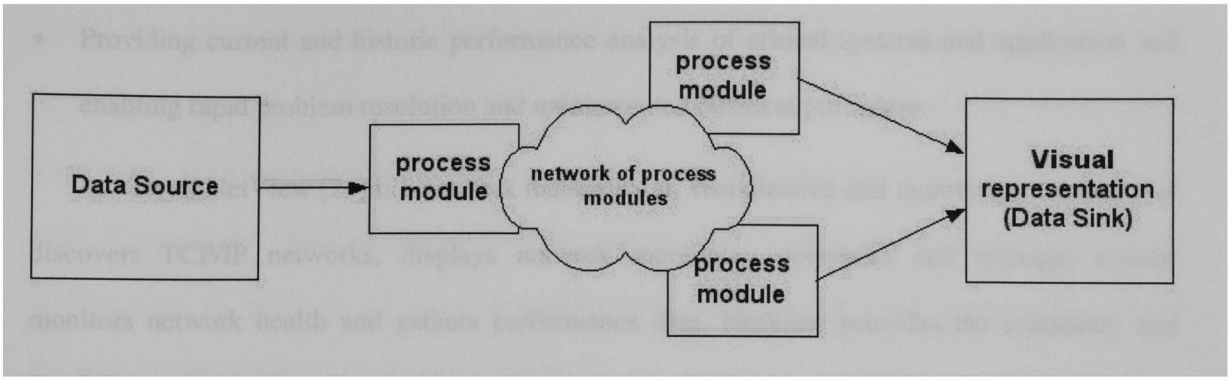

Figure 2. The Dataflow Paradigm

2.1.2 Conventional Monitoring and Performance Measurement Tools

Among the available tools [17] [18], [19], [20] Tivoli NetView and BMC Patrol are good examples of conventional monitoring and visualization tools. These tools have evolved without explicit concern for qualities like modularity, reusability and flexibility [2] in their designs. Due to such lack of concern the tools have evolved to be more domain-specific, thus restricting the applicability to only fields such as network management.

BMC Patrol is an enterprise management product line, where the sub products work together to provide monitoring and management for enterprise-wide distributed systems environments. The various services offered by BMC Patrol Management includes the following:

- Providing central point of control for all enterprise wide networks and communications devices with the ability to measure, monitor, communicate and improve availability and performance of the system.
- Detect various events, isolate and diagnose loss of service and report to central console.
- Predicting performance problems such as response time issues and allows users to pre-test solutions in performance management.

- Providing current and historic performance analysis of critical systems and application and enabling rapid problem resolution and uninterrupted business procedure.

Tivoli NetView [24] is a network management, visualization and monitoring solution that discovers TCP/IP networks, displays network topologies, correlates and manages events, monitors network health and gathers performance data. NetView provides the scalability and flexibility to manage and monitor large networks. NetView's scalability in enterprise level distributed network management allows configuration for network management of any scale.

2.2 RAD Framework and Assumptions for the Visualization System

This subsection explains the RAD authorization service and its architecture according tot the OMG CORBA specifications. The prototype for these specifications has already been developed at CADSE at FIU.

2.2.1 The RAD Framework

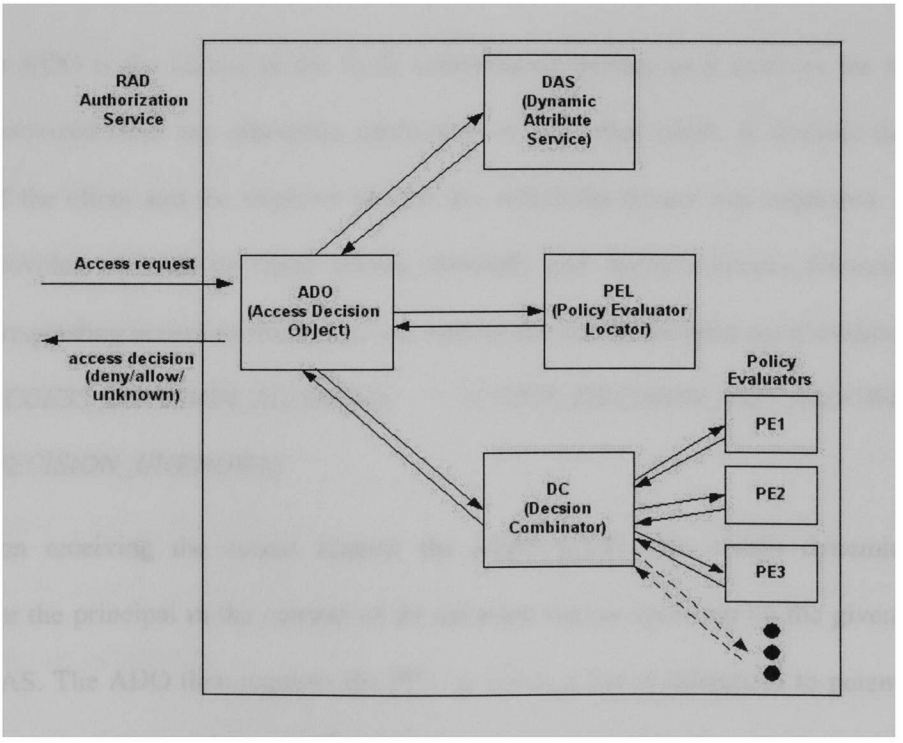

Figure 3. RAD Framework and Component interaction

The RAD authorization service is itself an implementation of the OMG CORBA facility [9] that goes by the same name, RAD (Resource Access Decider). It is important to obtain a good view of the RAD implementation in order to understand the data and metrics pertaining to the performance of its prototype. The reason being that the RAD framework has several components collaborating in order to provide the authorization service. The RAD framework and its component interaction are presented in Figure 3.

The components comprising the RAD framework are as follows:

1. Access Decision Object (ADO)
2. Dynamic Attribute Service (DAS)
3. Policy Evaluator Locator (PEL)
4. Policy Evaluator (PE)
5. Decision Combinator (DC)

The ADO is the facade of the RAD authorization service as it receives the request for access to resources from any enterprise application or any other client. It receives the security attributes of the client and the resource identity for which the access was requested. The ADO interface provides methods by name *access_allowed()* and *multiple_access_allowed()* for the purpose of requesting access to resources, and returns the access decision upon evaluation in the form *ACCESS_DECISION_ALLOWED*, *ACCESS_DECISION_NOT_ALLOWED* or *ACCESS_DECISION_UNKNOWN*.

Upon receiving the access request the ADO would then obtain dynamic security attributes for the principal in the context of the intended access operation on the given resource, from the DAS. The ADO then requests the PEL to obtain a list of references to potential policy evaluators and decision combinator. The DC combines the results of evaluations made by the

policy evaluators into a final decision by resolving evaluation conflicts and applying combination policies. The policy evaluators can be considered as distinct authorities each encapsulating a different set of authorization policies, which dictate access to resources.

The ADO can be invoked by the application server, which simulates the idea of an application employing RAD for the authorization logic. Any access request will reach the application server initially, where the decision whether the RAD needs to be utilized for processing the request, is taken.

The performance measurement tool would have to run tests that would simulate the client that sends in access control requests to the application. The scenario is depicted in Figure 1.

2.3 Desired Properties of RAD Performance Measurement Tools

Based on the background information available from the RAD authorization service and various monitoring and performance measurement systems a list of desired properties can be derived. The desired properties include modularity, scalability, reusability and flexibility to support RAD implementations developed at CADSE. The significance of these properties within the RAD performance measurement tool is explained below.

Modularity is an essential property for performance measurement tools as it adds the capability to divide the process into logically functional subsystems performing the relevant operations such as data extraction from the RAD authorization service prototype, subsequent computation, analysis and graphical representation. These stages in the absence of an integrated tool will require a different application for each of the stages. Further modular design of a system would allow the user/developer to make system modifications and changes that are localizes and thus avoiding any complexity.

Scalability allows the tool to serve applications running on distributed environments of any scale, such as a small regional network or a large enterprise level network. This can be achieved by making the different components or may be modules of the system capable to interact in distributed environments irrespective of size. This is an essential capability that the RAD performance measurement tool should provide.

Reusability and flexibility allow easy modification to the system and so these properties enable the capability to create generic interface for gathering performance data from different components of the RAD implementation. All the above properties will make the performance measurement tools capable of serving different implementations of the RAD framework.

3. The DOVE Framework

3.1 DOVE Research at Washington University

The Distributed Object Visualization Environment (DOVE) [25] is a visualization and monitoring framework developed as a part of research project headed by Dr. Douglas Schmidt at Department of Computer Science at Washington University. It is the result of the attempt to develop a framework that supports monitoring and performance measurement of applications and network management services in heterogeneous distributed systems. DOVE was developed as an exemplar to illustrate how frameworks and development tools, such as CORBA services and Java, can be combined with patterns such as Observer-Observable pattern; to build monitoring and measurement tools in distributed environments. Having researched into a few conventional monitoring and measurement tools, the framework was designed with explicit concern for software qualities like modularity, reuse and flexibility. To this end DOVE uses Object Oriented techniques, design patterns, CORBA, Java and C++ for its component technologies.

The DOVE project taken up at the Washington University has the following objectives:

- Create Web-based communication management application tools using Java component technology and design patterns that can automatically generate visualizations of system and application-level information in a network environment.
- Develop a real-time framework based on TAO (The ACE ORB) [26] to monitor and control applications and network elements in large-scale, hierarchical networks with minimal effort by developers and administrators.

- Develop a platform-independent, persistent, and hierarchical Management Information Base that allows applications to store and retrieve name-value bindings efficiently in a distributed environment.

The DOVE framework was developed as a combination of three types of applications, which are network management tools, application steering and application performance measurement. The services realized are discovering services, monitoring and configuring devices on the network through data storage in case of network management tools, online monitoring and control of applications in case of application steering and graphical and analytical view of application performance in case of application performance measurement.

3.2 DOVE Architecture, Components and Properties

The DOVE architecture [2] supports monitoring of distributed or embedded systems by minimizing the amount of computing overhead in the monitored application that may be caused due to processing performance or status information and providing feedback to users and administrators. The DOVE architecture includes the following major functional components:

1. DOVE Application
2. DOVE Management Information Base (MIB)
3. DOVE Agent
4. DOVE Browser
5. DOVE Visualization Component

The DOVE Architecture with the major components is displayed in the block diagram shown in Figure 4. The diagram demarcates between the DOVE Application and the monitoring framework.

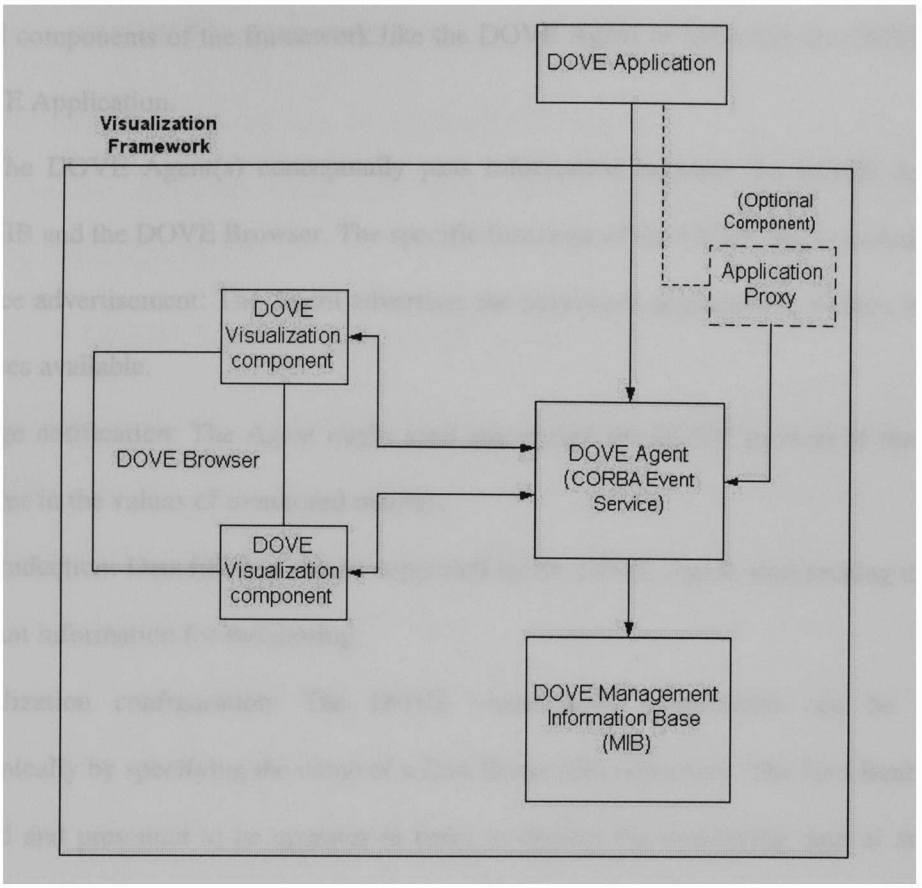

Figure 4. The DOVE Architecture and Components

The DOVE Application is the application that is to be monitored and measured for performance. The functionality of the DOVE Application is to publish information regarding its status and other metrics to DOVE Agents using optional components called the DOVE Application Proxies. The Application Proxy would only be an extension of the DOVE Agent with application specific functionality; thus negating any requirement of modifying the application to make it DOVE-enabled.

The DOVE MIB is a logical repository of information in the framework for storing the configuration, advertised services and other monitoring information pertaining to the monitored applications. All this information will be available for retrieval and can also be stored by other

interested components of the framework like the DOVE Agent or indirectly the DOVE Browser and DOVE Application.

The DOVE Agent(s) conceptually pass information between the DOVE Application, DOVE MIB and the DOVE Browser. The specific functions of the DOVE Agent include:

1. Service advertisement: The Agent advertises the monitored applications, metrics and control services available.
2. Change notification: The Agent might send any update the DOVE Browser if there are any changes in the values of monitored metrics.
3. Data reduction: Data filtering can be supported by the DOVE Agent, thus picking up only the relevant information for monitoring.
4. Visualization configuration: The DOVE visualization components can be integrated dynamically by specifying the name of a Java Beans [28] repository. The Java Beans are then loaded and presented to he operator in order to display the monitoring data of any type in different graphical forms.

The DOVE Browser serves as the front-end interface to the users. It provides the interface for the users to browse through the various monitored services and information from the target application. The Browser usually connects with one or more DOVE Agents in order to obtain the list of services and the pertaining monitoring information or data available. The user then selects the metrics to be displayed and the type of graphical format to be used for the representation.

The DOVE Visualization Component is conceptually a conduit between the end-user and the DOVE Application. Through the interface provided by the DOVE Browser the user could create and connect a visualization component that provides specific performance information and keeps track of events and updates related to this information.

3.3 DOVE Component Technologies

The DOVE framework can be implemented using the combination of technologies suggested by the concerned researching group at the Computer Science Department of Washington University [2], [27].

3.3.1 CORBA Event Service – DOVE Agent

The DOVE Agent is the most important component of the framework since it provides the capability for the performance measurement and monitoring system to operate in heterogeneous distributed environments. The CORBA Event Service [29] has been rightly selected for implementation as the DOVE Agent as it provides a generic manner in which data (called event) can be forwarded through distributed systems comprising of heterogeneous platforms, developing environments and applications. It provides a standard interface for interaction with either side of the data transfer process, the sender and the receiver sides, termed here as supplier and consumer respectively. The Event Service proves to be the scalable communication mechanism between DOVE components, providing efficient communication between possibly multiple DOVE Browsers and possible multiple DOVE Applications.

The Event Service can basically be used in two different ways, the push model and the pull model. In the push model, the supplier initiates the data transfer to the Event Channel while thereafter the Event Channel initiates transfer to the consumer. In the case of pull model the Event Channel initiates data transfer to itself from the supplier and the consumer initiates the transfer from the Channel to itself.

3.3.2 Java and DOVE Visualization Components

The DOVE Visualization Component should provide the feature of reusability, as it has to allow multiple representations of the data as per the performance measurement requirements.

The Java Beans [28], [30] are reusable software components and usually small control programs that can be visually manipulated in builder tools. The reusable software components can be simple push buttons, scroll bars, text fields or more complex components such as bar charts, graphical diagram editors, text editors, etc. The builder tools are nothing but the application building tools or more practically the GUI (Graphical User Interface) applications.

The Java Beans framework has been designed to allow objects to be written in such a way that their properties and behavior can be modified without having to recode or recompile existing components. The beans are Java classes that conform to certain standards and design guidelines [30], such as naming conventions for methods, etc.

With respect to the DOVE framework the DOVE visualization components could be implemented as Java Beans. The components will provide the tools required for the user to set the data and format for visualization. Based on the specific requirements of visualization there will be different components for different forms of data and representations as well. The design of the visualization components with the Java beans will be explained in the next chapter.

4. Modeling of the Performance Measurement Tool

The first step towards the implementation of the DOVE framework into a performance measurement tools or system involves detailed analysis and modeling. This chapter presents the requirement analysis and design modeling of the performance measurement tool implemented for testing of the RAD authorization service prototype. The tools used for modeling the system include UML (Unified Modeling Language) and Rational Rose. UML is a standard modeling language that defines notations and diagrams that aid in designing and representing a model for a system that is yet to be implemented.

4.1 Requirements Analysis

The basic objectives of the control flow visualization system the RAD authorization service as viewed from the highest level of abstraction, is to discover the service, obtain the relevant data and provide an interface through which the user could choose and display the data in the most perceptual visual forms.

4.1.1 Use case views and templates

The basic requirements at the high level view of the system are listed as follows:

1. The system should provide a list of the data fields from RAD, that are available for visualization display.
2. The system should also allow the user to select any of the data fields and map to any of the perceptual graphical representation forms.
3. Optionally, the system could also allow the user to save and store the data visualized.

These requirements have been modeled into a high-level use case diagram displayed in Figure 5. This use case diagram summarizes the basic functionality that the visualization system as a whole provides to the end-user. The DOVE components work together in order to satisfy these requirements.

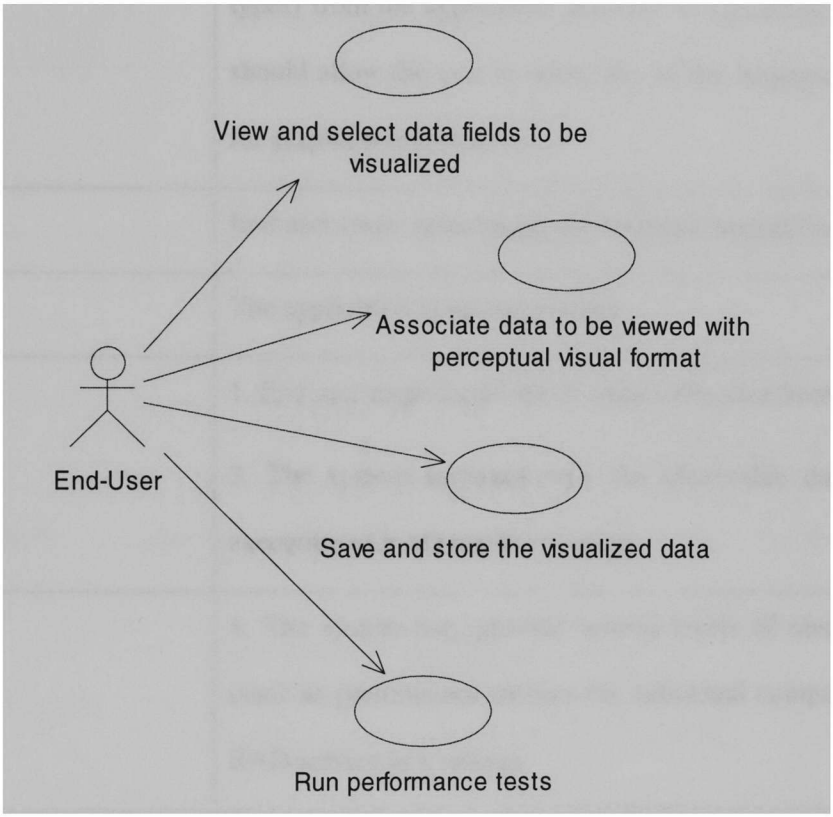

Figure 5. High level use case view of the Control Flow Visualization System

A detailed formal representation of each of the use cases is shown in the form of use case templates in Table 2, Table 3, Table 4 and Table 5 in the following pages.

Use Case	**#1. View and Select data to be visualized**
Description	The end-user should be able to view all the data (fields or types) from the application that can be visualized. The system should allow the user to select any of the displayed data fields for graphical representation.
Actors	End user (*role*: actor taking performance measurements)
Assumptions	The application is up and running.
Steps	1. End user requests the list of observable data from the system. 2. The system responds with the observable data from the execution of RAD implementation.
Variations	1. The system may provide several levels of observable data (such as performance metrics for individual components or the RAD service as a whole).

Table 2. Use case template for - view and select data

Use Case	**#1. Associate data to visual format**
Description	The end-user should be able to select the data.
Actors	End user (*role*: actor taking performance measurements)
Assumptions	1. The application is up and running. 2. Performance tests not yet started
Steps	1. End-user chooses the format. 2. The system responds with the observable data.
Variations	1. The system may provide a graphical format by default.

Table 3. Use case template for – selecting viewing format for data

Use Case	**#1. Save and store data**
Description	The end-user should be able to save the data.
Actors	End user (*role*: actor taking performance measurements)
Assumptions	1. Performance tests have been performed.
Steps	1. End-user chooses to save data and graphics.
Variations	1. The system must be able to reproduce the saved data and graphics.
Non Functional	Waiting Time: The service must be granted in reasonable time.

Table 4. Use case template for – save and store data

Use Case	#1. Run Performance Tests
Description	The end-user should be able run the tests using the tool.
Actors	End user (*role*: actor taking performance measurements)
Assumptions	1. Application server is up and running.
Steps	1. End-user enters test parameters using the performance measurement tool.
Variations	1. The system must be able to reproduce the saved data and graphics.
Non Functional	Waiting Time: The user must be granted service within reasonable time.

Table 5. Use case template for – run performance tests

4.1.2 Use cases of component interactions

Since the DOVE framework is being adopted for the implementation the interactions between the different DOVE components has to be modeled and the mutual requirements can be specified with the use case diagrams.

The RAD Authorization Service as the DOVE Application:

The RAD authorization service generates the performance data in response to the performance tests initiated by the end-user indirectly through this performance measurement system. RAD also communicates the performance data to the Event Channel (DOVE Agent)

either directly or indirectly. The use case diagram in Figure 6 and the templates in Table 6, Table 7 show the required functionality of the RAD towards performance measurement system.

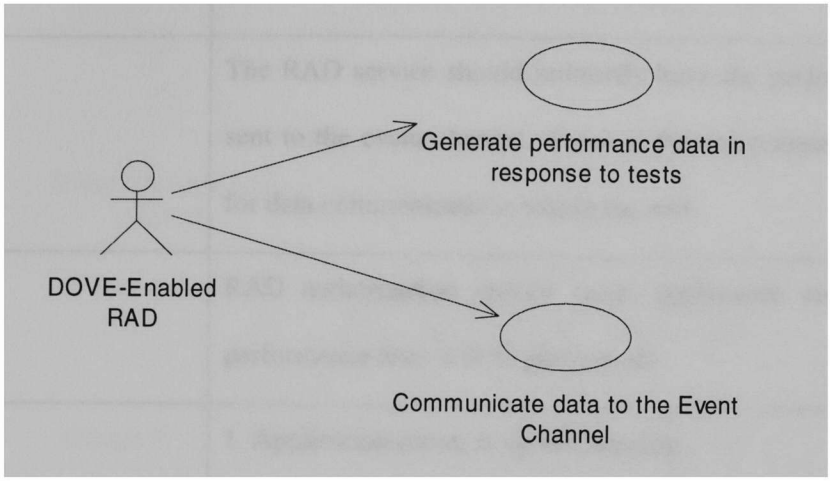

Figure 6. DOVE-Enabled RAD Use Case Diagram

Use Case	**#1. Generate performance data in response to tests**
Description	The RAD authorization service should generate data when the end-user indirectly runs performance tests on the service.
Actors	RAD authorization service (*role*: application on which the performance tests will be performed)
Assumptions	1. Application server is up and running.
Steps	1. Generate performance data according to the requirements.
Variations	1. The service need not really be aware of the presence of the performance measurement tool.
Non Functional	RAD need not prioritize this against its application logic.

Table 6. Use case template for – Generate performance data

Use Case	#2. Communicate data to the Event Channel
Description	The RAD service should indirectly have the performance data sent to the event channel, which is the component responsible for data communications within the tool.
Actors	RAD authorization service (*role*: application on which the performance tests will be performed)
Assumptions	1. Application server is up and running.
Steps	1. Create entities that are responsible to send the data to event channel without obstructing the RADs own authorization logic.
Variations	1. The RAD service need not really be aware of the presence of the performance measurement tool.
Non Functional	RAD need not prioritize this against its application logic.

Table 7. Use case template for – communicate data to event channel

The Event Channel as the DOVE Agent:

The Event channel is the performance measurement system's equivalent of the DOVE Agent. The basic requirements include the following: receiving performance data from the RAD authorization service, reporting the data to the end-user indirectly. The data will be consumed by the components that register with the channel on behalf of the end-user's invocations and performance tests. Thus the channel is a conduit between the RAD service and the end-user interface components of the performance measurement tool. The requirements are modeled into the use case diagram in Figure 7 and the use case templates shown in Table 8 and Table 9.

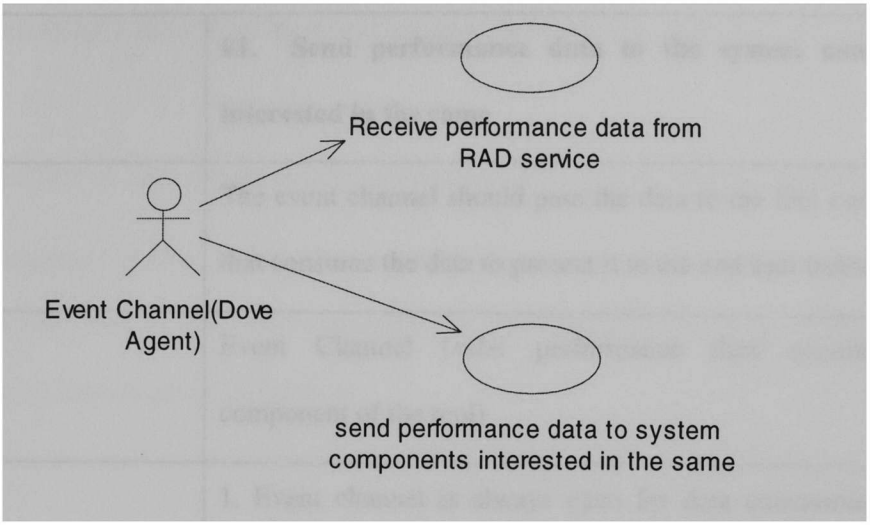

Figure 7. Use case diagram of the Event Channel

Use Case	**#1. Receive data from RAD authorization service**
Description	The event channel should accept any number of connections from RAD service to receive all the performance data requested by the end-user indirectly through the tool's interface.
Actors	Event Channel (*role*: performance data communication component of the tool)
Assumptions	1. Application server is up and running.
Steps	1. Send the data to event channel without obstructing the RAD's own authorization logic.
Variations	1. The event channel can employ context-based strategies in the process.

Table 8. Use case template for – receive data from RAD

Use Case	**#1. Send performance data to the system components interested in the same**
Description	The event channel should pass the data to the tool components that consume the data to present it to the end user indirectly.
Actors	Event Channel (*role*: performance data communication component of the tool)
Assumptions	1. Event channel is always open for data communication on both receiving and sending sides.
Steps	1. Record the components interested in receiving the data. 2. Send the data to such components.
Variations	1. The event channel can employ context-based strategies in the process.

Table 9. Use case template for – sending performance data

DOVE Browser – End User Interface

The DOVE Browser is the end user interface component of the performance measurement tool. It is responsible to coordinate all the interactions on the receiving end of event channel and present the performance data to the end user. This component is responsible for the following actions:

1. Provide interface for the end-user to run performance tests.
2. Create visualization components to display performance characteristics to the end-user.

3. Receive performance data from the event channel and feed the updates to the appropriate viewers.

4. Provide capability to save and store performance data and graphical data for the end-user.

This component is also capable of monitoring the behavior of the RAD components and represents it in the form of a control flow diagram. The functionality of this component is displayed in the use case diagram in Figure 8.

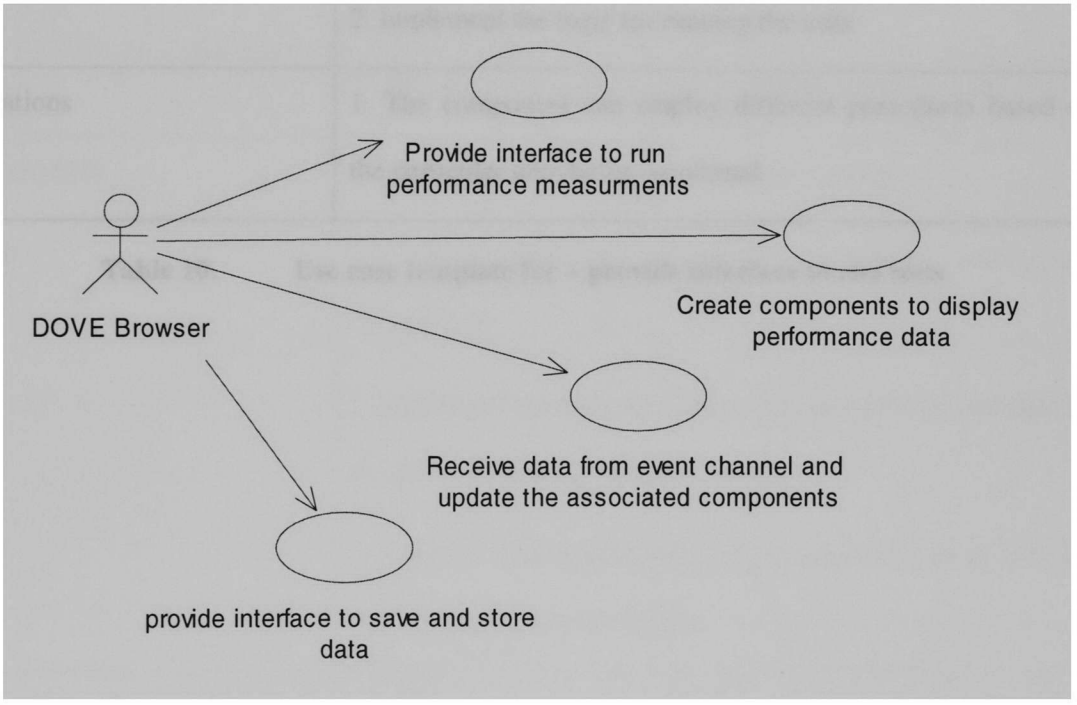

Figure 8. DOVE Browser use case diagram

The step-by-step execution of each use case is described by the use case templates shown in Table 10 and Table 11 in the following pages.

Use Case	**#1. Provide interface to run performance tests**
Description	The browser should represent an interface to the end-user to run the tests.
Actors	DOVE Browser (*role*: end-user interface and data distribution)
Steps	1. Provide user interface to run tests 2. Implement the logic for running the tests.
Variations	1. The component can employ different procedures based on the particular item being monitored.

Table 10. Use case template for – provide interface to run tests

Use Cases	**#2. Create components to display performance data** **#3. Receive data from event channel and update the associate components** **#4. Provide interface to save and store performance data**
Description	The browser creates individual components to deal with a individual performance-monitoring requests
Actors	DOVE Browser (*role*: end-user interface and data distribution)
Assumptions	1. The browser is aware of all other components.
Steps	1. Allow end-user to create individual visualization components. 2. Implement the logic for these components to receive data on behalf of the user and update the view. 3. Cater to other requirements of the user, such as to save the data and reproduce the results.

Table 11. Use case template for – create and update visualization components

The Visualization Component – as a supplement to the Browser component:

The visualization component is required to register with the Event Channel for any updates to the data it is associated with by the end-user and propagate the update to the visual representation. The related use case diagram is indicated in Figure 9 and the use case template already presented previously in Table 10 and Table 11.

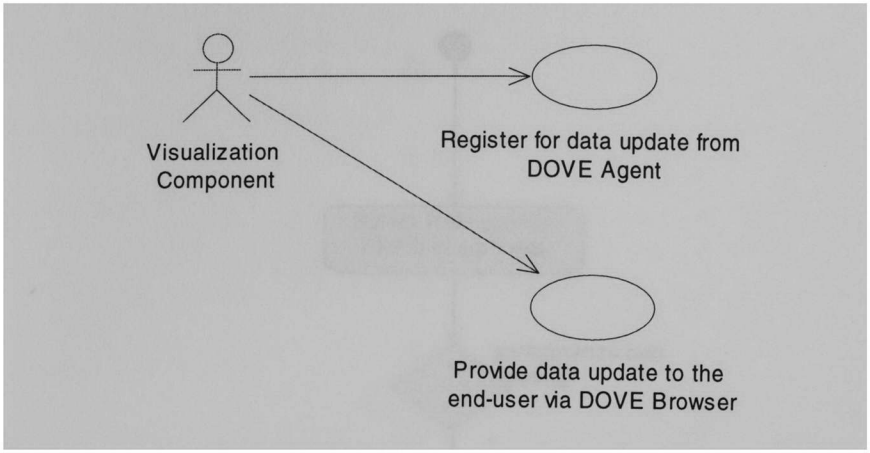

Figure 9. Use case diagram for DOVE Visualization Component

4.1.3 Activity Diagrams modeling utility of the system

This subsection discusses the analysis stage modeling of the system's utilities towards the end-user.

The utility of the performance measurement tool/system can be modeled in the form of the UML activity diagrams. The two basic operations the end-user could carry out with the performance measurement tool are invoking the performance measurement tests and then observing the RAD service or RAD component(s) behavior/performance during the tests. These diagrams are displayed in Figure 10 and Figure 11.

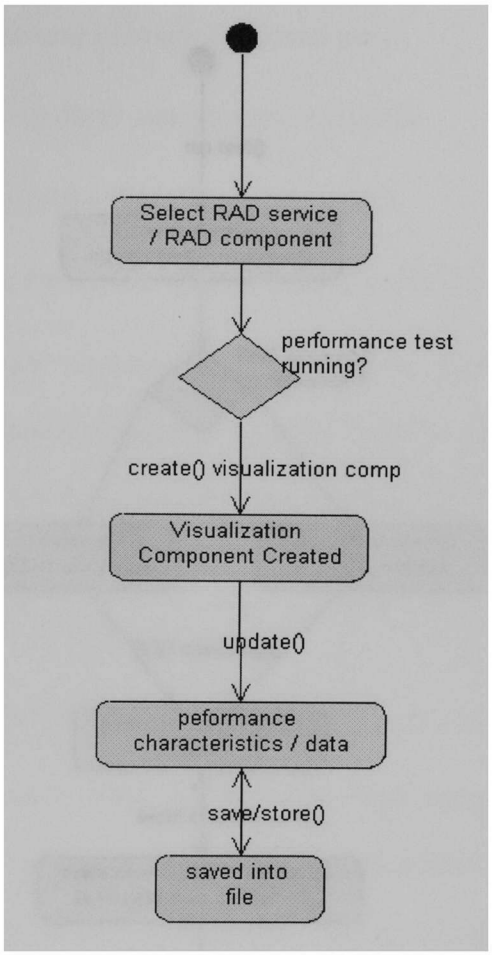

Figure 10. Activity diagram indicating the performance measurement test

These sequence diagrams identify the sequence of events that make up the whole process. The activity diagrams identify the interactions/invocations between the different components of the performance measurement tool. They indicate important information needed in developing further design model diagrams such as the sequence diagram and the class diagram, which would aid in completing the system model.

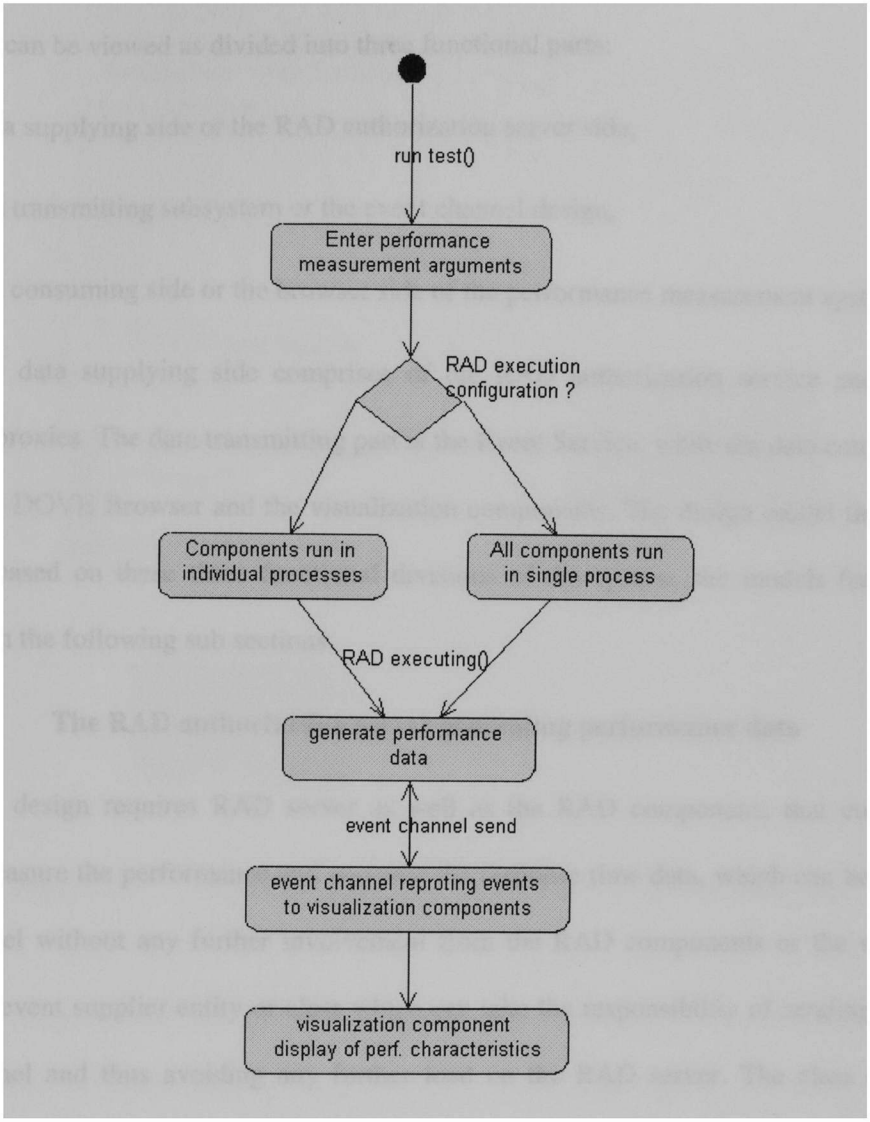

Figure 11. Activity diagram for running performance measurement tests

4.2 Design model of the Performance Measurement Tool

The design model for the performance measurement tool will explain the detailed functionality and construction of the system. The model has been represented in the form of class diagrams and sequence diagrams derived from the analysis model presented in previous subsection and the DOVE architecture itself.

The design can be viewed as divided into three functional parts:

1. The data supplying side or the RAD authorization server side,
2. the data transmitting subsystem or the event channel design,
3. the data consuming side or the browser side of the performance measurement system.

The data supplying side comprises of the RAD authorization service and any other supporting proxies. The data transmitting part is the Event Service, while the data consuming side includes the DOVE Browser and the visualization components. The design model thus has been developed based on these three functional divisions of the system, the models for which are elaborated in the following sub sections.

4.2.1 The RAD authorization server generating performance data

The design requires RAD server as well as the RAD components that constitute the server to measure the performance and generate the response time data, which can be sent to the event channel without any further involvement from the RAD components or the server. This calls for an event supplier entity or class which can take the responsibility of sending the events to the channel and thus avoiding any further load on the RAD server. The class diagram so obtained at this design level is shown in Figure 12.

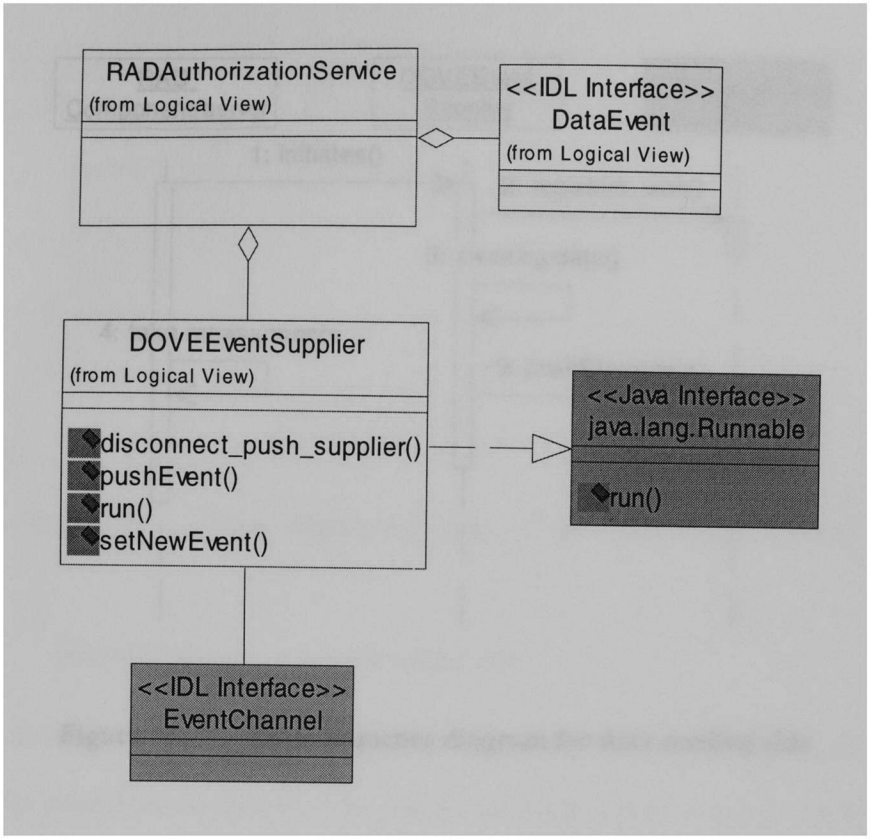

Figure 12. Basic class diagram of RAD server data sending side

The design shows the *DOVEEventSupplier* class, which is responsible to continuously push the events available from the RAD server or component through to the event channel component of the system. This class employs a software thread that listens for new data events and sends them when available. The interaction is better depicted in the sequence diagram shown in Figure 13.

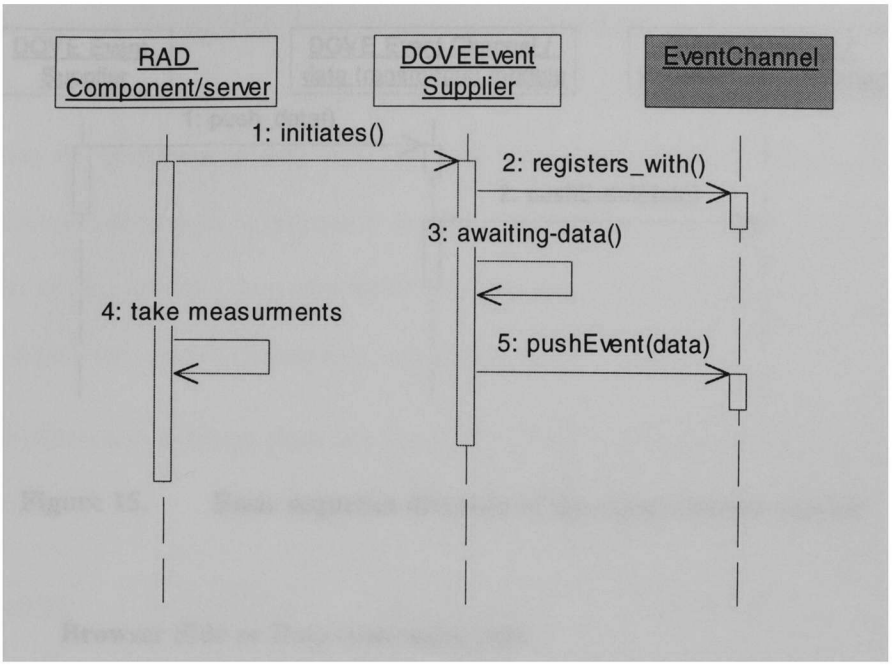

Figure 13. Basic sequence diagram for data sending side

4.2.2 The Event Channel or the data transmitting component

This component has a very straightforward but vital functionality of reading in data from the RAD server side and pushing the data to the browser side towards the visualization components that request for the related data. The simple design is displayed in the class diagram in Figure 14 and the sequence diagram shown in Figure 15.

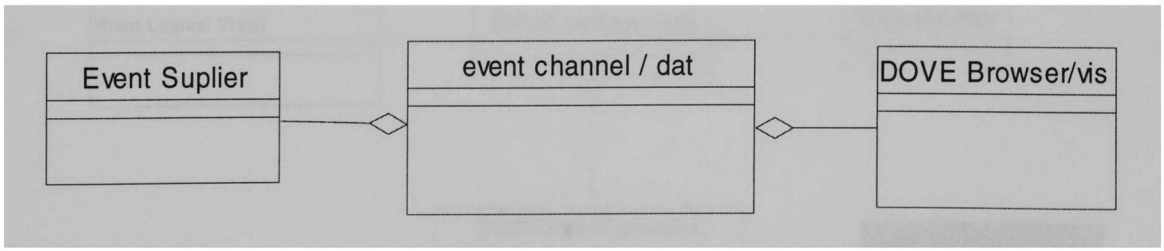

Figure 14. Basic class diagram of event channel module

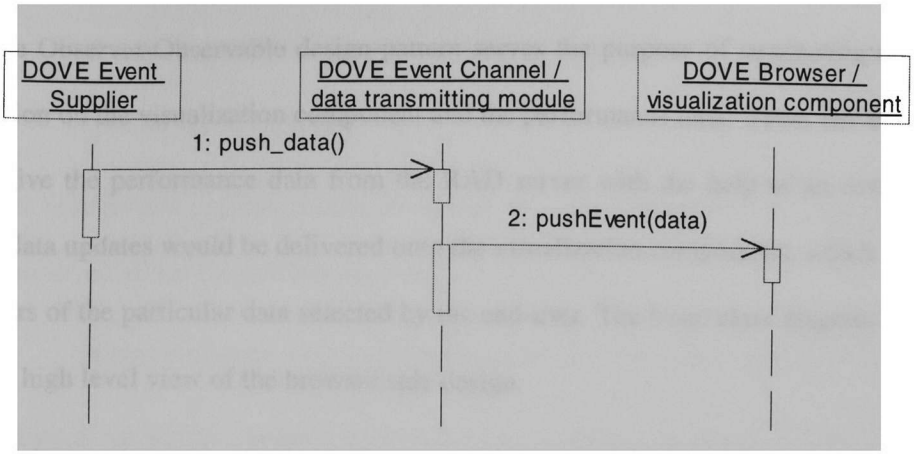

Figure 15. Basic sequence diagram of the event channel module

4.2.3 Browser Side or Data consuming side

The Browser side design of the performance measurement tool requires the browser to coordinate the input from the end-user through the graphical interface and the performance data input from the event channel.

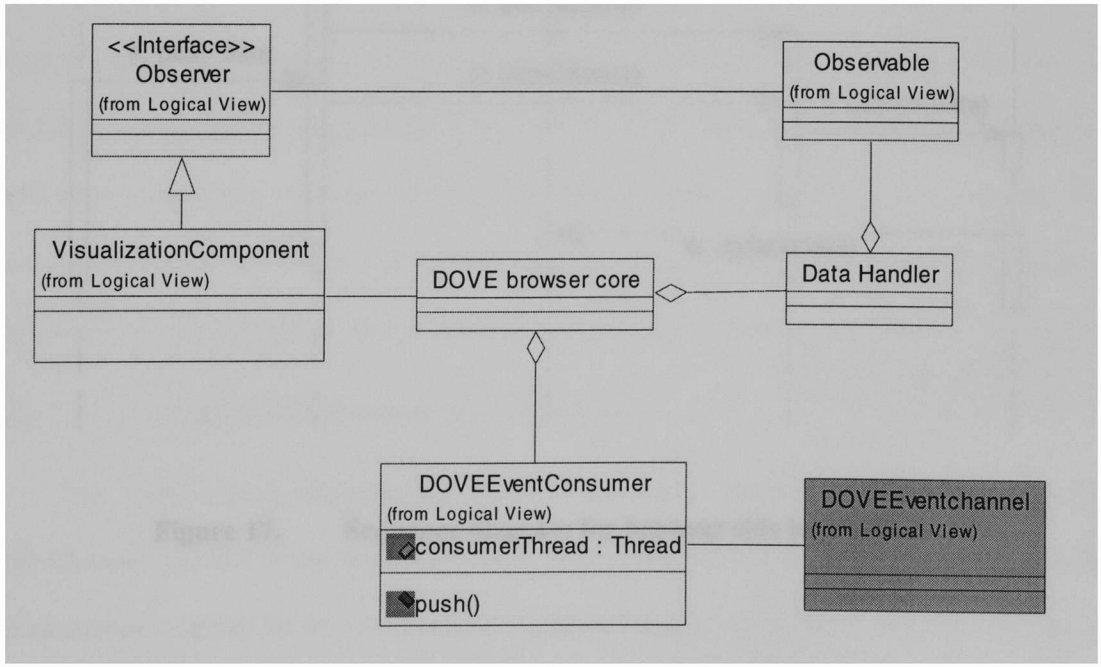

Figure 16. Class diagram for browser side design

The Observer-Observable design pattern serves the purpose of connecting the graphical representation on the visualization component and the performance data. Thus, the browser class would receive the performance data from the RAD server with the help of an event consumer class. The data updates would be delivered onto the visualization components, which would act as the observers of the particular data selected by the end-user. The basic class diagram in Figure 16 depicts this high level view of the browser side design.

The interaction between these classes is described with the help of a sequence diagram shown in Figure 17.

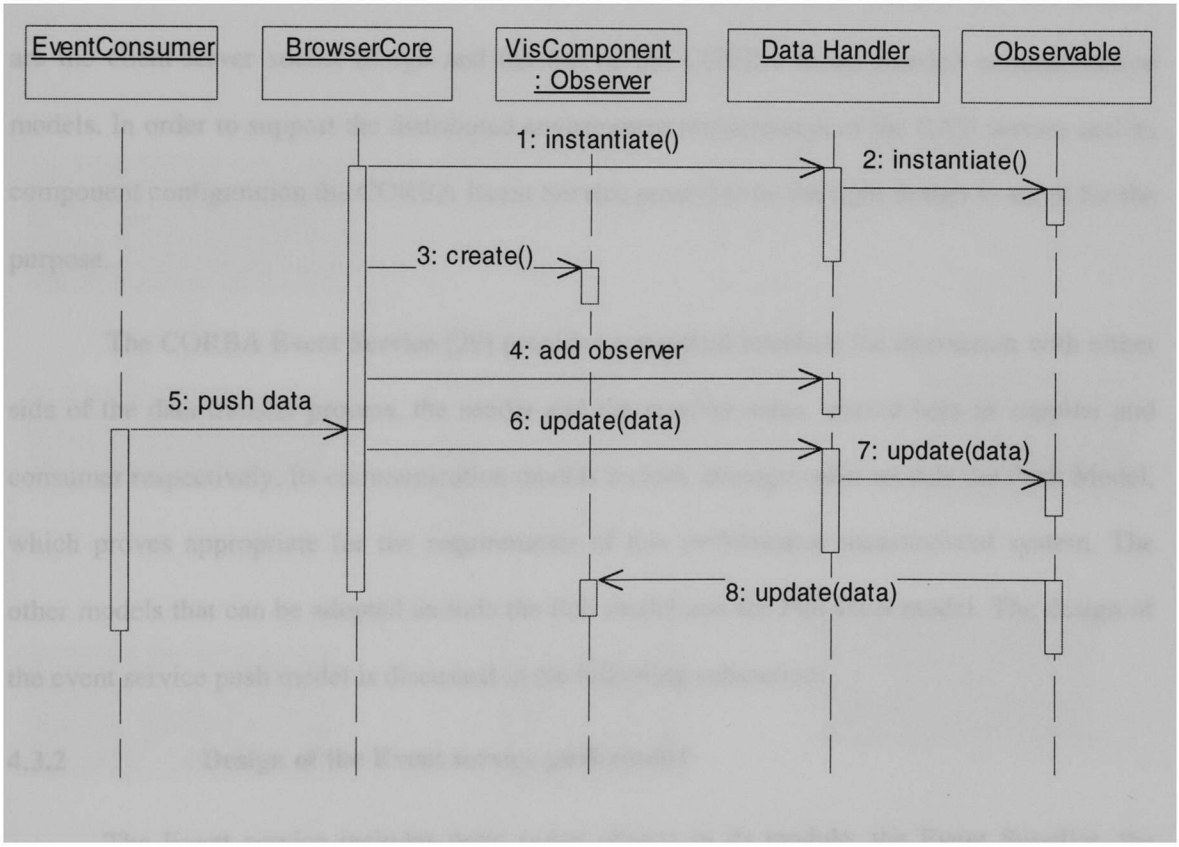

Figure 17. Sequence diagram for browser side interaction

4.3 Comprehensive implementation model for the performance measurement system

This section presents the details filled into the basic design level to complete the final implementation model for the performance measurement tool.

4.3.1 Design decisions taken for the final model of Performance measurement tool

The most important decision to be made to complete the design model is that of the communication mechanism, which is responsible to transfer the performance data between the RAD service and the browser module of the performance measurement system. The two choices are the client-server socket design and the use of the CORBA Event Service communication models. In order to support the distributed environment requirements of the RAD service and its component configuration the CORBA Event Service proved to be the right design to adopt for the purpose.

The CORBA Event Service [29] provides a standard interface for interaction with either side of the data transfer process, the sender and the receiver sides, termed here as supplier and consumer respectively. Its communication models include amongst other models the Push Model, which proves appropriate for the requirements of this performance measurement system. The other models that can be adopted include the Pull model and the Pull-Push model. The design of the event service push model is discussed in the following subsection.

4.3.2 Design of the Event service push model

The Event service includes three major objects in its module, the Event Supplier, the Event Channel and the Event Consumer. The push model of the Event service [29] has the communication initiated by the side sending/supplying the data or the Event supplier. The data is

"pushed" to the event channel and the event channel further pushes the data to Event Consumer(s) that have registered with the event channel to receive data.

The data is communicated in generic format using the CORBA.Any [29] structure, which is capable of encapsulating any kind of data. In terms of the design model, the push model involves the *PushSupplier,* the *PushConsumer* interfaces and the *EventChannel.* The RAD service starts the *PushSupplier* during initialization so that the supplier would wait on the data and push it to the channel when the performance data is available.

4.3.3 Design of the data supplying side

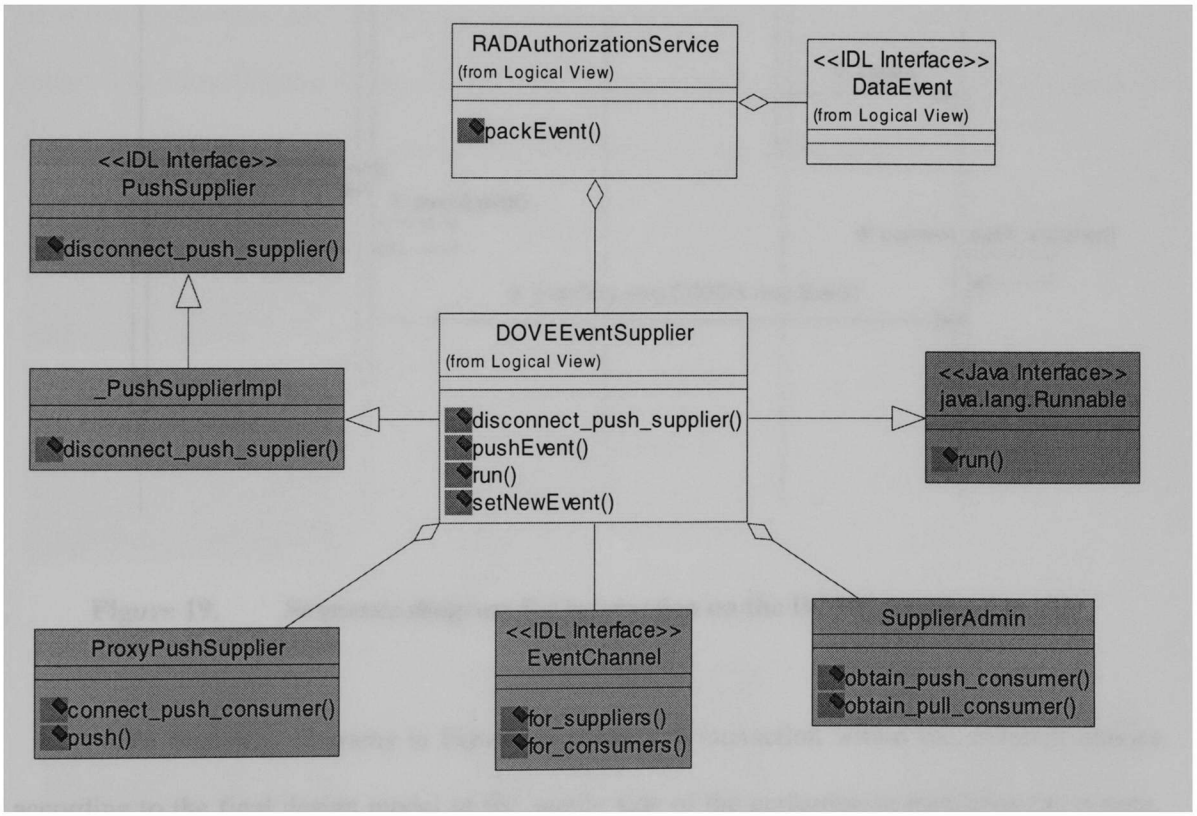

Figure 18. Class diagram of final design of the Supplier side

The performance-data supplier will be an inheritance of the implementation of the *PushSupplier* interface named as the class *DOVEEventSupplier*, which has the functionality to

push events through the event channel and disconnect the push supplier from the event channel. Further, the *DOVEEventSupplier* also implements the Java *Runnable* interface to continuously supply the events from the RAD server. The detailed class diagram for the supply side (RAD authorization side) is presented in Figure 18.

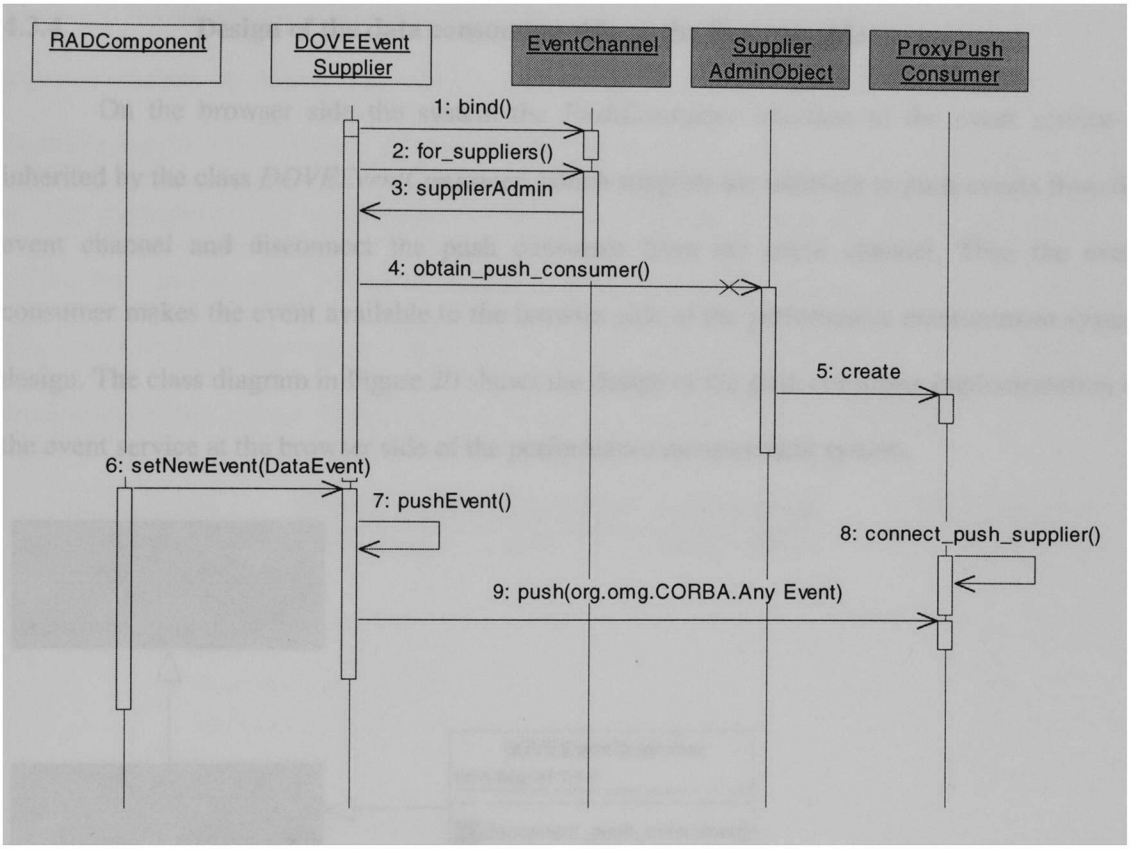

Figure 19. Sequence diagram for interaction on the DOVE Application side

The sequence diagrams in Figure 19 shows the interaction within the different classes according to the final design model of the supply side of the performance measurement system. The *DOVEEventSupplier* object binds to the *EventChannel* and calls the *for_suppliers()* method in its interface to obtain the *SupplierAdmin* object handle. The *SupplierAdmin* object is then prompted to create a *ProxyPushConsumer* object that operates from within the event channel providing the interface to push events through the channel. The *ProxyPushConsumer* also

connects the supplier to the event channel by calling the connect_push_supplier() method which takes the *DOVEEventSupplier* as the parameter. In order to dispatch an event through the event channel the connected supplier would call the *push()* method supported by the *ProxyPushConsumer*.

4.3.4 Design of the data consuming side or the Browser side

On the browser side the system the *PushConsumer* interface of the event service is inherited by the class *DOVEEventConsumer*, which supports the interface to push events from the event channel and disconnect the push consumer from the event channel. Thus the event consumer makes the event available to the browser side of the performance measurement system design. The class diagram in Figure 20 shows the design of the push consumer implementation of the event service at the browser side of the performance measurement system.

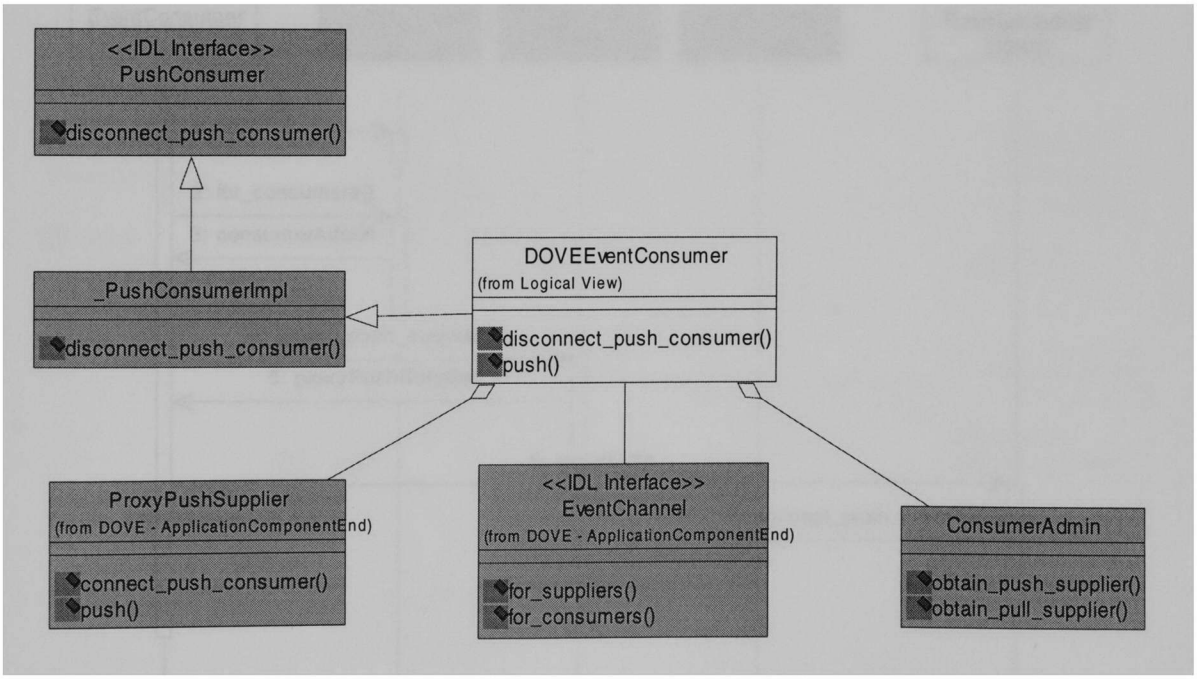

Figure 20. Class diagram of the Event consumer design on the browser side

The event consuming side of the performance measurement system where the Browser side receives the event updates involves the interactions regarding event consumption, as shown in the detailed sequence diagram in Figure 21. The *DOVEEventConsumer* object binds to the *EventChannel* and calls the *for_consumers()* method in its interface to obtain a handle to the *ConsumerAdmin* object. The consumer then obtains the *ProxyPushSupplier* object from the *ConsumerAdmin* by calling the *obtain_push_supplier()* method. The *ProxyPushSupplier* is the interface that the event channel provides to interact with the consumer. The *PushConsumer* object is then instantiated and is connected to the event channel through the *ProxyPushSupplier* by calling the *connect_push_consumer()* method, which takes the consumer object as its parameter. The connected *PushConsumer* object now actively offers the services of its push method to receive any event updates from the event channel through the *ProxyPushSupplier*.

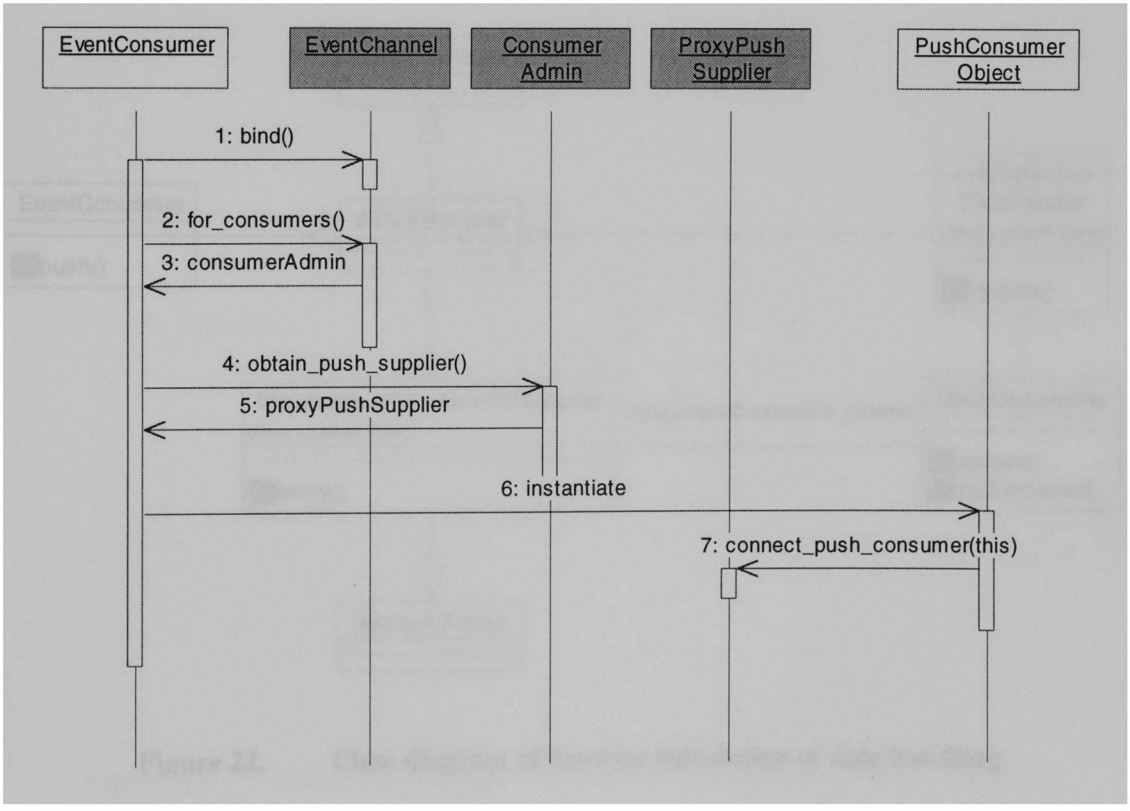

Figure 21. Sequence diagram showing detailed browser side event consumption

4.3.5 Design of the Browser interaction and interface

The DOVE browser side of the system implements the *Observer-Observable* design pattern [2], [3] in order to support the associations between the performance data and the user interface. The observer pattern allows notification to interested clients upon any changes to the data being observed. The pattern defines two categories of objects called the Observer and the Observable. The *Observers* register with one or more of the *Observables*. An *Observable* informs the *Observers* about any changes as soon as its own state changes. When applied to the Browser side of this performance measurement system the metrics/data is encapsulated in the *Observables* and the Visualization components are the represented as the *Observers*. The browser will be the object that receives the updates from the event consumer on the browser side and connects the associated *Observable* to the appropriate *Observers* or Visualization Components.

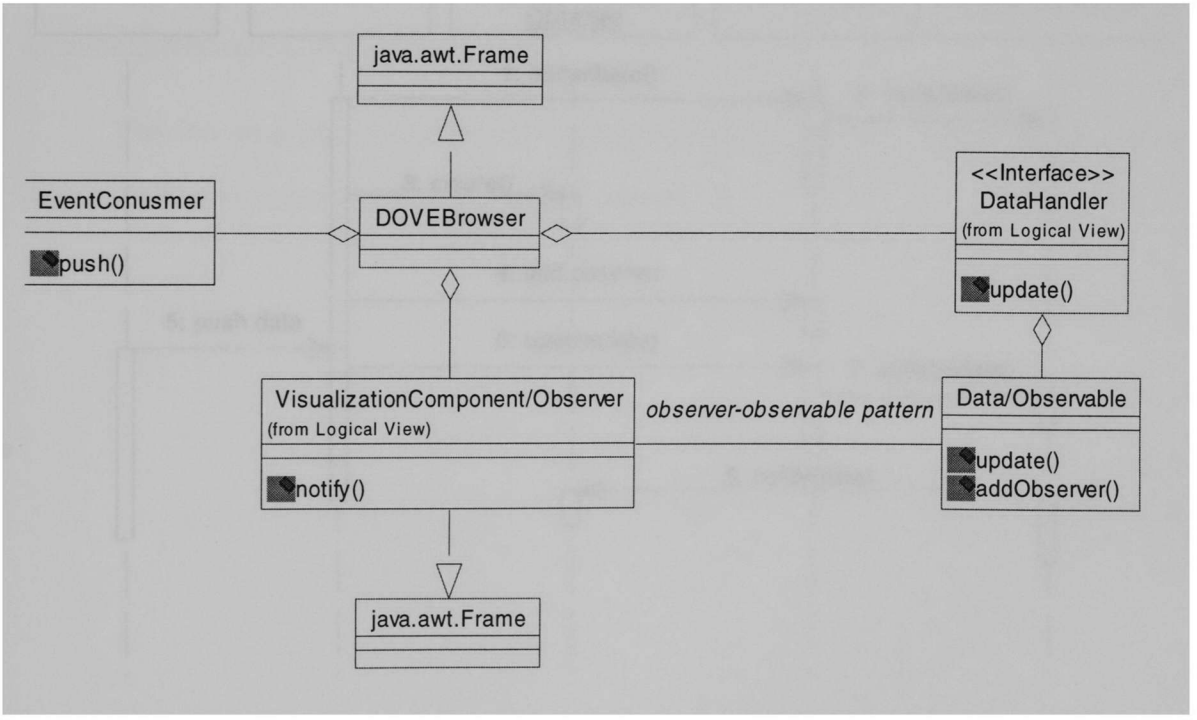

Figure 22. Class diagram of browser side design of data handling

The class diagram shown in Figure 22 displays the design of the data handling on the browser side. The details of the interaction between the different classes involved in the Browser side are described by the sequence diagram in Figure 23. The browser is also the host for the user interface where the user chooses the *Observable* to be displayed. The browser core object then instructs a visualization component factory to create a visualization component with properties necessary to display the chosen *Observable*. The data update events are forwarded to the Data handler, to be more concrete the Time data handler. This handler demultiplexes the event structure into several metrics. The metrics are the *Observables*, which notify the events to the *Observers*. The Observers (visualization components) upon reading the notification update the graphical display on the front-end.

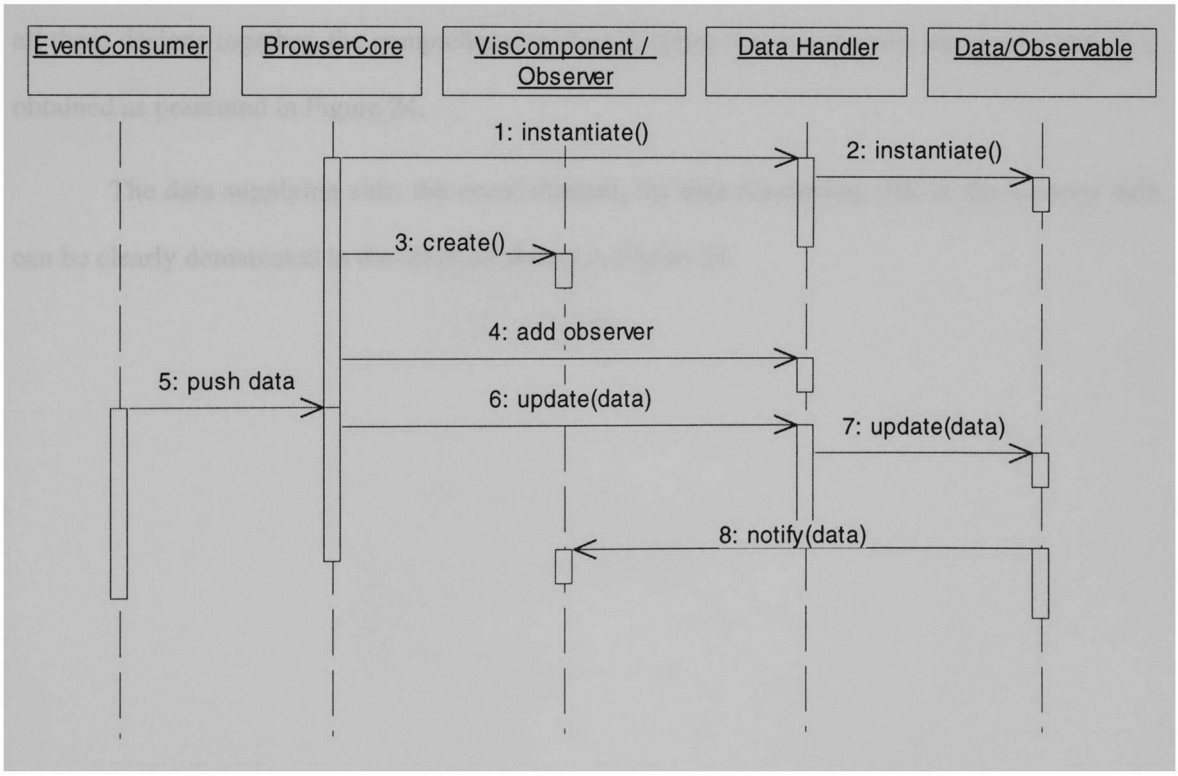

Figure 23. Sequence diagram of interactions in the browser data handling

The *BrowserCore* is the bridge between the user and the performance data, instantiates the *DataHandler* which processes the events specifically dealing with data. The *BrowserCore* also initiates the *EventConsumer* object and passes on the data handler object to it. Upon receiving the event update the *EventConsumer* forwards the event to the *DataHandler* which then extracts the data from the event and instantiates the appropriate *Observable*, in this case the *Observable* and sends the update to it. The *Observable* further notifies the *Observer* or the *VisComponent* (a visualization component).

4.3.6 Complete design model

The design model presented till the previous section presents the design and interactions within the component sub systems or modules of the performance measurement tool. Integrating all these designs together, the comprehensive class diagram that summarizes the system design is obtained as presented in Figure 24.

The data supplying side, the event channel, the data consuming side or the browser side can be clearly demarcated in the diagram shown in Figure 24.

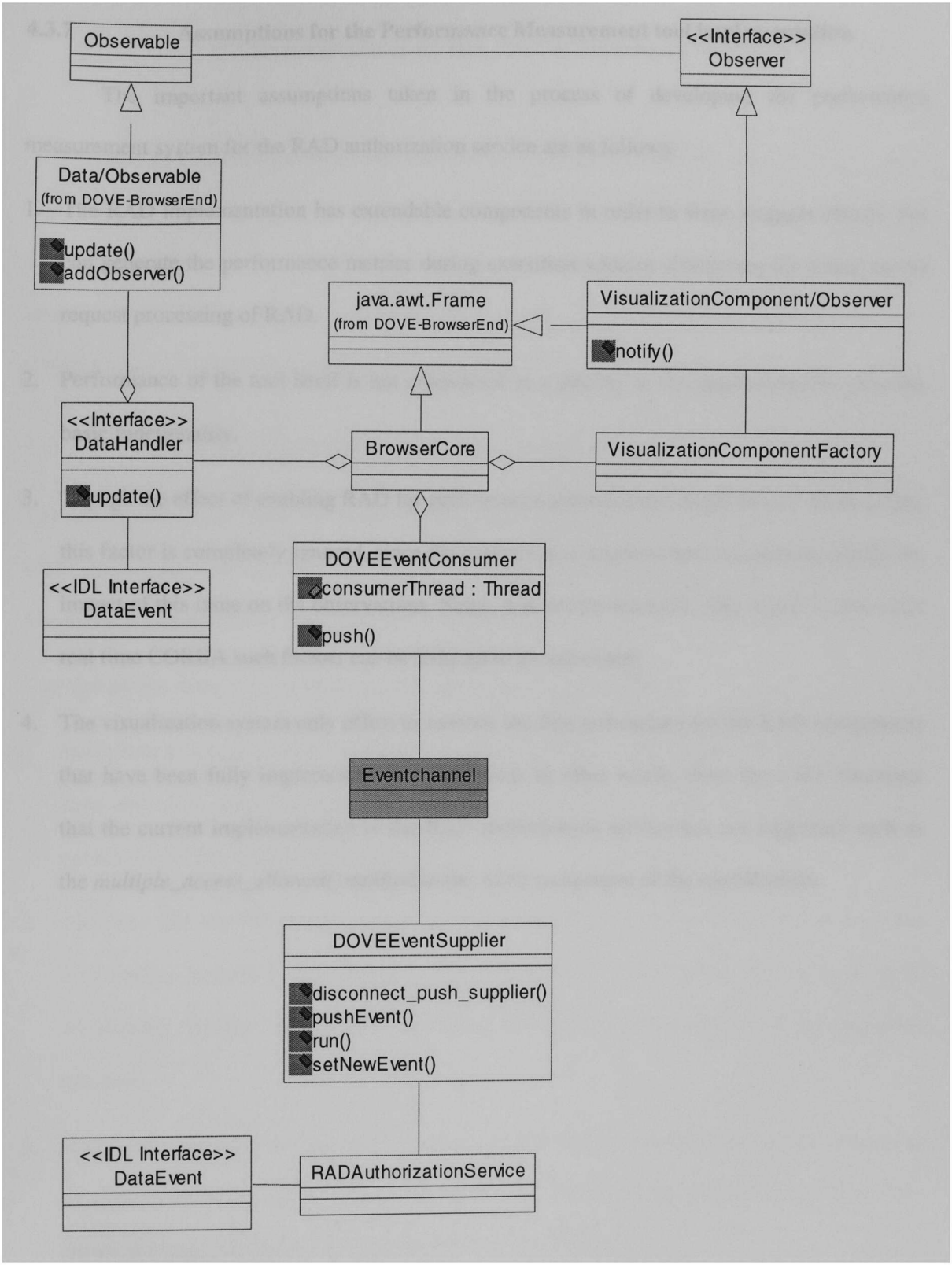

Figure 24. Comprehensive class diagram of the system model

4.3.7 Assumptions for the Performance Measurement tool implementation

The important assumptions taken in the process of developing the performance measurement system for the RAD authorization service are as follows:

1. The RAD implementation has extendable components in order to write wrapper classes that can generate the performance metrics during execution without obstructing the actual access request processing of RAD.

2. Performance of the tool itself is not considered as a priority as the implementation provides basic functionality.

3. Though the effect of enabling RAD for performance measurement might slow it down a little, this factor is completely ignored, since the performance measurement calculations nullify the impact of this issue on the observations. Since, it is known from [2], [26], and [27] that using real time CORBA such factors can be reduced to the minimum.

4. The visualization system only offers to monitor the data and metrics for the RAD components that have been fully implemented and functional. In other words, there are a few functions that the current implementation of the RAD authorization service has not supported such as the *multiple_access_allowed()* method in the ADO component of the specification.

5. Implementation of the Performance Measurement Tool

This chapter explains about the implementation of the performance measurement system. The decisions made regarding developing tools used for the implementation, the presentation of the user interface of the visualization system and explanation of the integration of the visualization system to a more available and distributed implementation of RAD called the CORBA-based Application Authorizaton Service (CAAS) are covered in respective sections.

5.1 Development Tools used for Implementation

The development tools used for the implementation of the control flow visualization system for RAD authorization service include Java programming language [32] over the JBuilder Client Server Suite and VisiBroker for Java [4] for CORBA programming.

The reasons for using Java as the programming language are as follows:

1. Java presents simple and easy programming style. The RAD implementation was also done in Java and so it makes development easier and simpler in enabling the performance data generation by the RAD authorization server.

2. The Java API for GUI programming is very resourceful in developing the front end of the visualization system (in the Browser and Visualization components). The Java Abstract Windowing Toolkit (AWT) [32] has been used for the development of the graphical interface.

3. Java further provides the option of developing the front-end visualization system browser as an application or an applet, which can be viewed on any of the web browsers. Thus Java allows for wide availability of the system on the Internet too.

The reasons of using JBuilder Client Server version for Java programming are:

1. The implementation requires the use of CORBA ORB and the Event Service implementation, which are available in the form of basic VisiBroker ORB implementation included with the JBuilder suite.

2. In order to have a common development tool for the application as well as the visualization tool JBuilder seemed the right option. The RAD authorization service was also developed using the same tool and thus guides through well approved styles of coding and documenting for the visualization system development.

Since the RAD authorization service has been developed to optionally use CORBA Object Request Broker (ORB) for communication over distributed environments. VisiBroker for Java ORB implementation had been used for this purpose. Since the visualization system also requires the CORBA Event Service implementation for the DOVE Agent VisiBroker proves to be the right choice.

5.2 User Interface of the Performance Measurement System

The front-end of the performance measurement tool is hosted by the Browser component, which uses Java AWT Frames to present the GUI components that control the performance measurement and monitoring process. The browser offers different interfaces for conducting the performance measurement tests on the RAD servers and viewing the results form the performance measurement tests.

5.2.1 Performance test execution interface

The interface developed for the end-user to execute performance tests on the RAD authorization server takes several test arguments/parameters as the input from the user. Theses arguments include certain configuration file(s) to boot up the RAD components, the set of RAD

components to be initialized, the configuration of the RAD components and other program options set by the RAD developers. The different configurations in which the RAD server can be run are to run all the components in a single process or to run the individual components in respective individual processes. A screen shot of the frame that shows the interface is displayed in Figure 25.

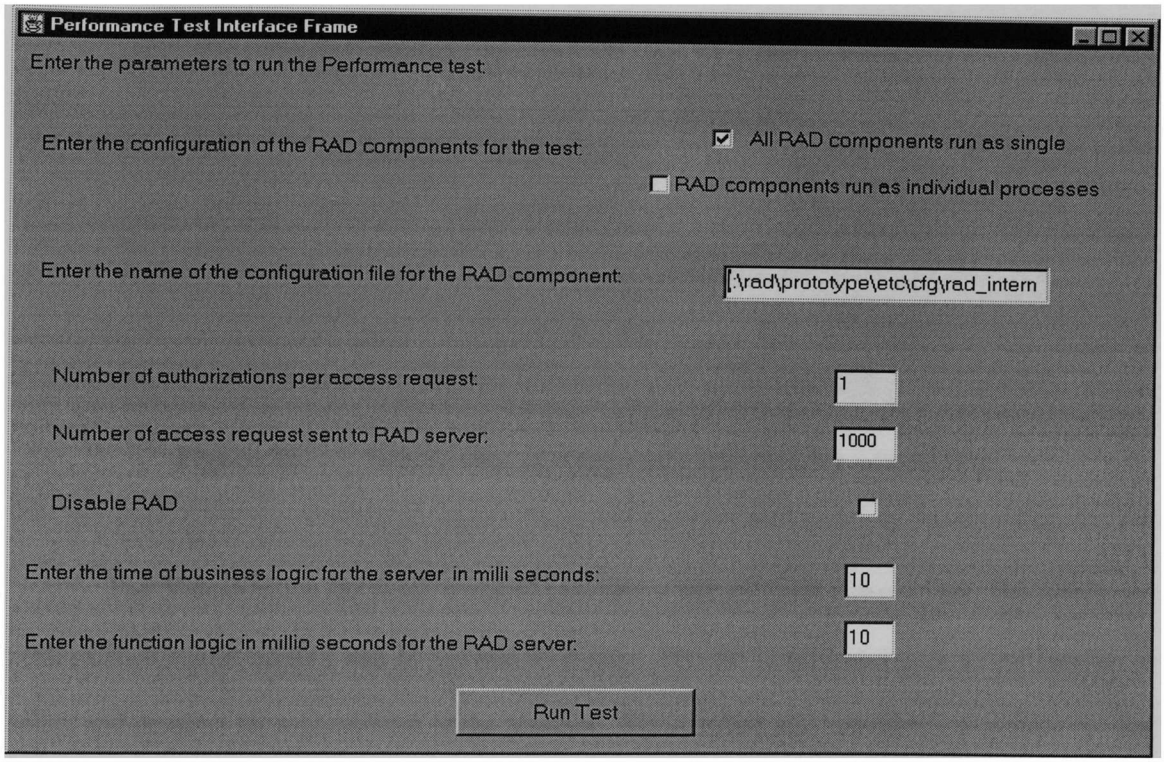

Figure 25. Screen shot of the Performance testing interface

The text field for number of authorizations takes the input for the number of times each request is to be repeated by the RAD server. The field for number of access requests takes in the value for the number of access requests to be sent to the server. An input to disable RAD is also provided in order to run the tests without using the RAD server, wherein the server would not contact the ado for access requests but process the request by itself. Other parameters include

business logic and functional logic time to be entered in milliseconds. The performance tests can be executed by clicking on the "Run Test" after having entered valid input.

5.2.2 Performance measurement viewing interface

The performance characteristics can be viewed by interacting with a simple interface offered by the Browser. The screenshot of the interface is shown in Figure 26.

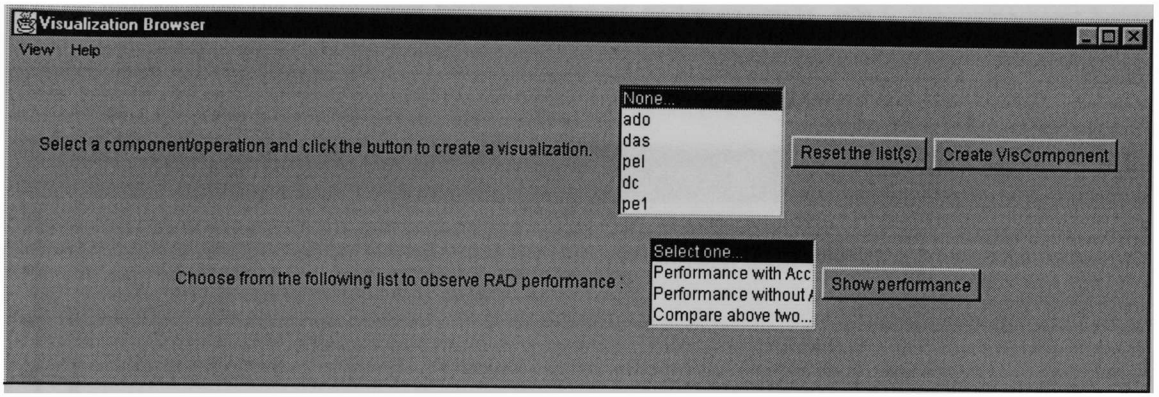

Figure 26. Screen shot of component selection for monitoring

The Visualization Browser window provides a list of RAD components, for which the performance characteristics can be viewed over time. The list is followed by a set of buttons to select and deselect for repeated use of the window. The interface also provides a selection list that gives a choice between viewing the performance of the RAD server itself when the RAD logic or disabled. Further the choice allows the end-user to compare the performance when access control is enabled, to the performance when access control is disabled. This set of choices is followed by a button that would start the visualization component frame, which opens its own interface windows to the end-user to view and manipulate the performance data and characteristics that have been requested for.

The next stage of interaction in the process will be the Visualization component frame, which is run by the component that is genuinely created for processing the particular component.

5.2.3 Visualization component interface

The visualization component creates the performance characteristics showing the response time of the candidate RAD component (ADO, DAS, PEL, DC or PE) or the RAD server over time in milliseconds. The interface also includes button controls to save/file the data and recollect on demand. A snapshot of this window is shown in Figure 27. The title of the window will be named after the component being monitored, as for example, ado is the component being monitored as per the Figure 27.

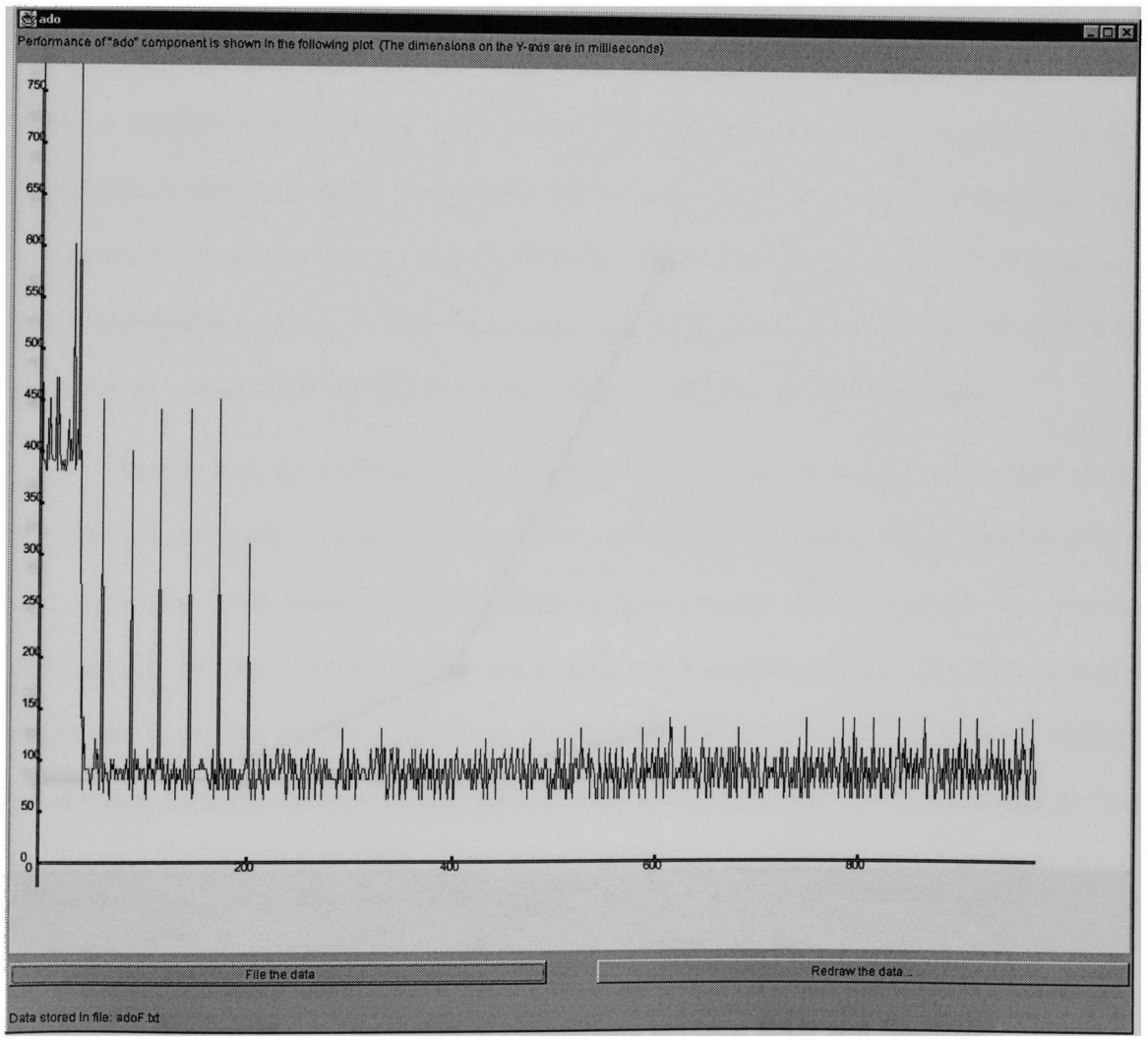

Figure 27. Snapshot of visualization component frame for ado performance

The visualization component is also capable of indicating the performance of the complete RAD server with respect to its average response time for 1, 10, 100 and 1000 access requests. The component will also provide a comparison with the case where the authorization server does not invoke/use the RAD logic or any RAD components to return. Such a comparison is a good estimation of the performance of the RAD, as the comparison is made with a scenario where the application server is completely bypassing it.

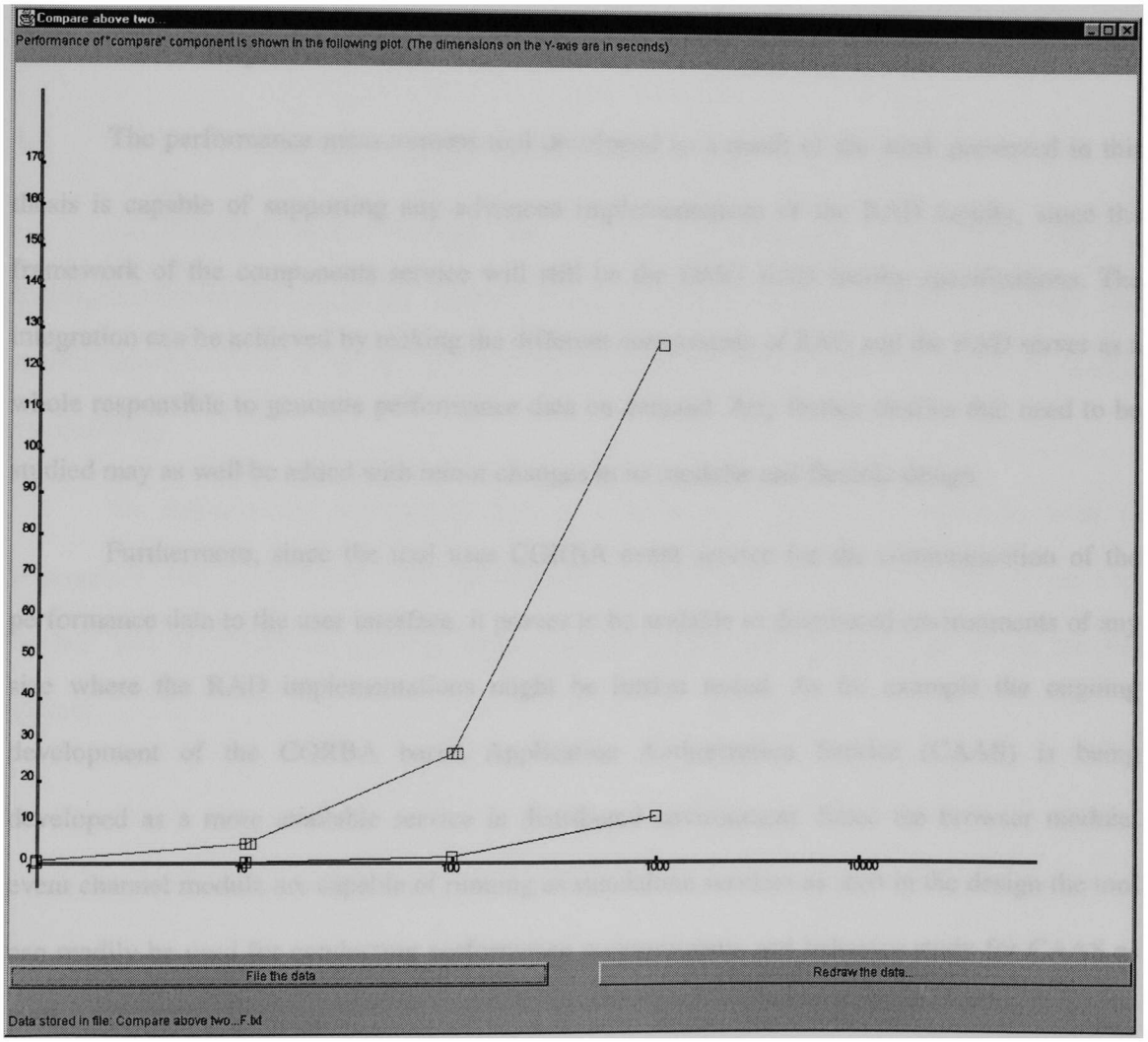

Figure 28. Snapshot of comparison between RAD and No-RAD

A snapshot of the comparison drawn by the performance measurement tool for the RAD service prototype is shown in Figure 28. The bottom plot in the figure shows the performance (response time) when the RAD implementation is bypassed and the upper plot indicates the performance when the RAD is requested to process the access control request without being bypassed.

5.3 Integration of the Performance Measurement Tool with other RAD Implementations

The performance measurement tool developed as a result of the work presented in this thesis is capable of supporting any advanced implementations of the RAD facility, since the framework of the components service will still be the OMG RAD facility specifications. The integration can be achieved by making the different components of RAD and the RAD server as a whole responsible to generate performance data on demand. Any further metrics that need to be studied may as well be added with minor changes in its modular and flexible design.

Furthermore, since the tool uses CORBA event service for the communication of the performance data to the user interface, it proves to be scalable to distributed environments of any size where the RAD implementations might be further tested. As for example the ongoing development of the CORBA based Application Authorization Service (CAAS) is being developed as a more available service in distributed environment. Since the browser module, event channel module are capable of running as standalone services as seen in the design the tool can readily be used for conducting performance measurements and behavior study for CAAS as well.

6. Conclusions

The work presented in this thesis includes a performance measurement system for the RAD authorization service prototype developed at CADSE, FIU. The following sections will present the comparison matrix introduced in the first chapter in subsection 1.4, Table 1, followed by some major contributions of the thesis, applicability of the work and a perspective of what future work needs to be taken up or can be taken up.

6.1 Evaluation and Comparison matrix with the existing process

As described before through this report the performance measurement tool has highlighted the integration of the different stages of the process into a single tool/system. The matrix shown in Table 12 clearly identifies and explains the upper hand of the tool over existing methodology.

To elaborate the comparison further, it takes large amount of effort in order to accumulate data from about 1000 authorization requests and analyze and interpret the data further. This is exactly the problem faced with the un-automated methodology of the existing system.

Properties	Existing method for performance measurement	Performance Measurement Tool/System
Time overhead / resources requirement	Requirement of different applications and more time for the following stages: data collection time, data computation, analysis and plotting	Reduction in effort and resources needed. Data collection, computation and plotting are all achieved by one integrated system.
Data	Limited performance data is available for the authorization service as a whole	Data is available for individual components, apart from the authorization service as a whole.

Table 12. Final performance matrix of the tool against existing method

The samples of the results obtained by either method are shown in Figure 29 and Figure 30.

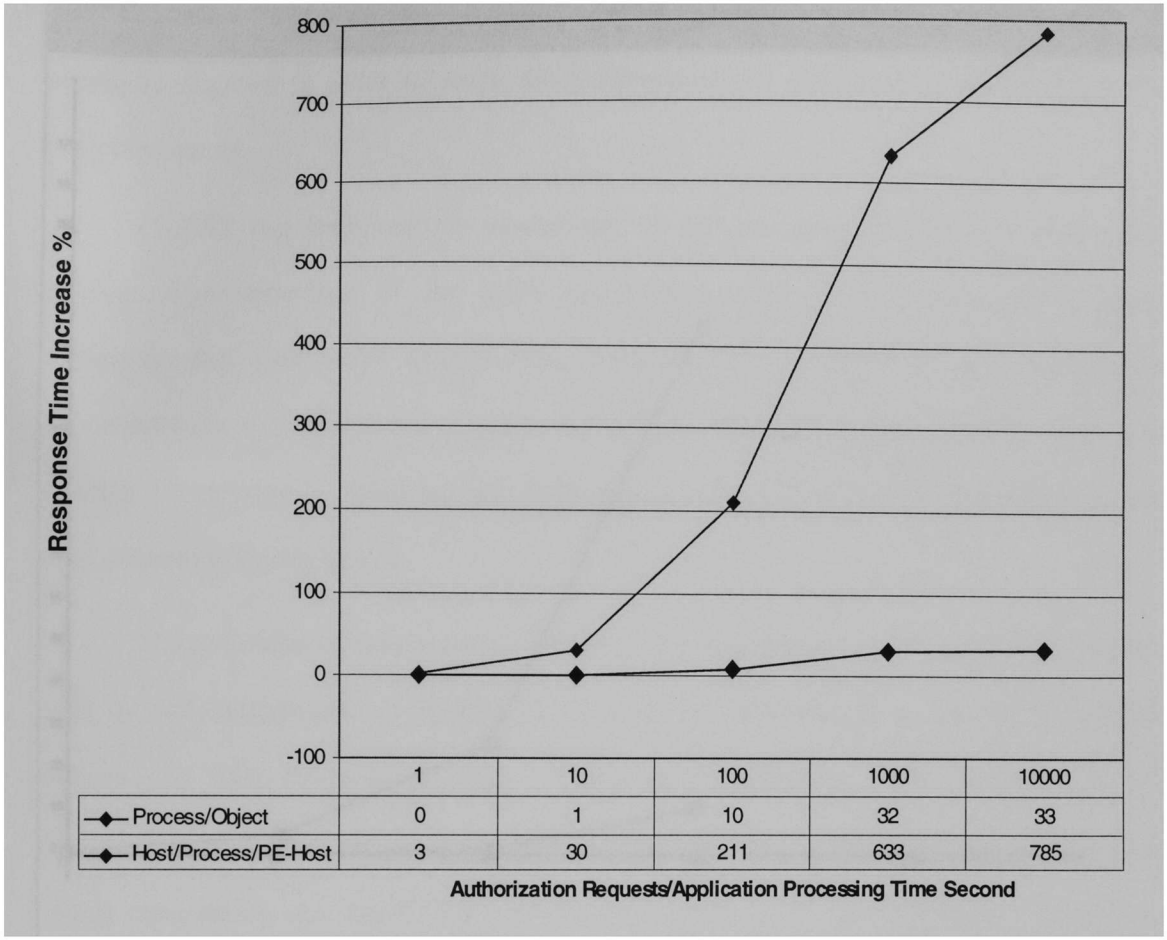

Figure 29. Performance characteristics as obtained by existing methodology

It can be observed that there isn't a noticeable change in the quality of the results obtained, thus verifying the basic ability of the performance measurement tool.

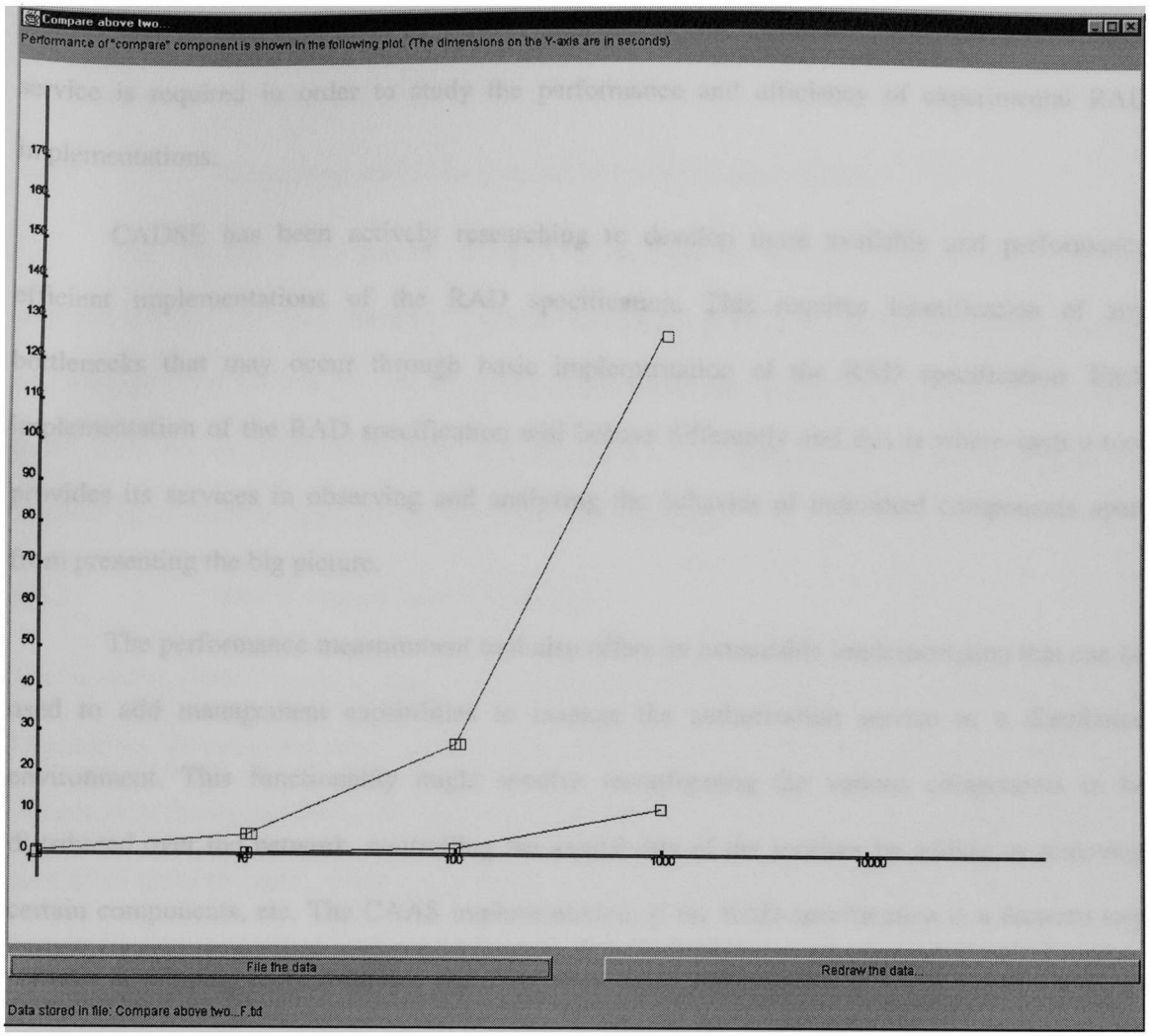

Figure 30. Performance characteristics as obtained from the tool

6.2 Contributions of the work

6.2.1 Contributions of the tool towards RAD implementations

The major contribution of the work is the implementation of the performance measurement system that measures the performance and monitors the control flow and other metrics of the RAD authorization service. The system is an implementation of the DOVE framework developed at Washington University. The visualization of resource consumption

metrics such as time consumed by various components and operations within the authorization service is required in order to study the performance and efficiency of experimental RAD implementations.

CADSE has been actively researching to develop more available and performance efficient implementations of the RAD specification. This requires identification of any bottlenecks that may occur through basic implementation of the RAD specification. Each implementation of the RAD specification will behave differently and this is where such a tool provides its services in observing and analyzing the behavior of individual components apart from presenting the big picture.

The performance measurement tool also offers an extendable implementation that can be used to add management capabilities to manage the authorization service in a distributed environment. This functionality might involve reconfiguring the various components to be distributed over the network, controlling the availability of the services by adding or removing certain components, etc. The CAAS implementation of the RAD specification is a forward step towards developing more available and distributed RAD authorization services and this tool can readily put to used for the testing.

6.3 Applicability of the performance measurement system

The performance measurement tool/system is applicable to monitoring other metrics in RAD-based implementations.

6.3.1 RAD-based Authorization Systems

However since the RAD framework which has been monitored through this system is a well-designed and modular architecture, the visualization system can be enhanced to monitor and manage other similar distributed applications. The degree to which the data is visualized can vary

from fine-grained to coarse-grained based on the application components and corresponding functionality.

6.3.2 Monitoring and Management of Multiple Applications

The tool has a very generic interface for enlisting the viewable aspects of applications' data and metrics. The only change that would be required in order to adapt the system to monitor and manage other applications would be the performance data generation enabling, which needs to be done on the application side. The system uses its modular framework to support the monitoring of multiple applications over distributed systems.

6.3.3 NMVC Component Technology

The scalable DOVE-based system has already been incorporated into an integrated Network Monitoring, Visualization and Control system (NMVC) for large and high-speed networks at the Washington University. The basic purpose of the NMVC system is to ensure adequate quality of service to network users, while maintaining high network resource utilization. The role of the system in such integrated large scale systems will be to manage the quality of service (QoS) provided by network components and allowing end-users and applications the capability to apply dynamic feedback about their realized QoS to help control applications and system resources more effectively.

6.4 Future Work

The following list indicates prospective development areas for the performance measurement tool presented in this thesis:

1. CORBA-based Application Authorization Service (CAAS) Performance visualization and monitoring: The system can be scaled to visualize the CAAS, which as explained before is a more available implementation of the RAD facility.

2. Addition of Management capability: With the capability of the system to utilize both the Push and the Pull models of the CORBA event service, data can be exchanged between the user interface and the application being monitored. This capability will enable the application management service of the system. The system can thus be enhanced in order to build an application monitoring as well as management system, which would be more versatile than the present implementation.

3. Real time performance measurement system implementation can be achieved with the use of the Real time CORBA Object Service Event Channel implementation and The Ace ORB (TAO) [26], both developed at Washington University. Such a system would provide additional capabilities to study the application behavior in real time. Further Real time Event service implementation is considered to reduce the visualization computation load on the application side.

List of References

[1] Konstantin Beznosov, Yi Deng, Bob Blakley, Carol Burt and John Barkley, "A Resource Access Decision Service for CORBA-based Distributed Systems", Proceedings of the Annual Computer Security Applications Conference, Phoenix, Arizona, 1999.

[2] Michael Kircher, Douglas Schmidt, "DOVE: A Distributed Object Visualization Environment", Appeared in C++ Report, SIGS, Vol. 11, No 3, March, 1999.

[3] Bill McCarty, Luke Cassady-Dorion, "Java Distributed Objects", Sams Publishing, 1999.

[4] Visigenic Training Services, "High Performance Distributed Applications with VisiBroker for Java", Visigenic Software, Inc., Copyright 1997.

[5] Allan Tuchman, David Jablonowski, George Cybenko, "Run-time Visualization of Program Data", Proceedings of IEEE conference on Visualization, held in San Diego, California, 1991.

[6] A Wierse, U Lang, R Ruhle, "Architectures of Distributed Visualization Systems and their Enhancements", Eurographics Workshop on Visualization in Scientific Computing, Abingdon, 1993.

[7] BMC Patrol, http://www.bmc.com/patrol, BMC Software, 2000.

[8] Philip Robertson and Lisa De Ferrari, "Systematic Approaches to Visualization: Is a Reference Model Needed?", Scientific Visualization, Advances and Challenges, Edited by: L. Rosenblum, R.A. Earnshaw, J. Encarnacao, H. Hagen, A. Kaufman, S. Klimenko, G. Nielson, F. Post, D. Thalmann , Academic Press, 1994.

[9] Object Management Group, "Resource Access Decision Facility", OMG document number: corbamed/99-05-04, 1999.

[10] Luis Espinal, Konstantin Beznosov, Yi Deng, "Design and Implementation of Resource Access Decision Server", A Technical Report, 2000-01.

[11] Ravi Sandhu, "Access Control: The Neglected Frontier", Proceedings of the First Australasian Conference on Information Security and Privacy, Wollongong, Australia, 1996.

[12] John Barkley, Konstantin Beznosov, Jinny Uppal, "Supporting Relationships in Access Control Using Role Based Access Control", Proceedings of the fourth ACM workshop on Role-based access control, Fairfax, VA, USA, 1999

[13] A. Wierse, U. Lang, R. Rhule, "Architectures of Distributed Visualization Systems and their Enhancements", Eurographics Workshop on Visualization in Scientific Computing, Abingdon, 1993.

[14] R. B. Haber and David A. McNabb, "Visualization Idioms: A Conceptual Model for Scientific Visualization Systems", Visualization in Scientific Computing, Pages 74-93. IEEE Computer Society Press. 1990.

[15] Joe Bergin, Ken Brodie, Marta Patiño-Martínez, Myles McNally, Tom Naps, Susan Rodger, Judith Wilson, Michael Goldweber, Sami Khuri, Ricardo Jiménez-Peris, "An overview of visualization: its use and design", Proceedings of the conference on Integrating technology into computer science education, ITiCSE '96, ACM SIGCSE Bulletin Vol. 28, Pages 192-200, 1996.

[16] Ben Shneiderman, "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations", Proceedings of IEEE Symposium on Visual Languages '96, IEEE, Los Alamitos, CA, Pages 336-343, 1996.

[17] The Numerical Algorithms Group, "IRIS Explorer User's Guide", The Numerical Algorithms Group Ltd, Oxford, UK, 1995.

[18] Craig Upson, T Faulhaber, D Kamins, D Laidlaw,D Schlegel, J Vroom, R Gurwitz, and A Van Dam., "The Application Visualization System: a computational environment for scientific visualization," IEEE Computer Graphics and Applications, July, 1989.

[19] AVS 5 Overview, http://www.avs.com/products/AVS5/avs5.htm, Advanced Visual Systems, Inc. (AVS), 2000.

[20] Jeremy Walton, "Data Visualization with IRIS Explorer – What's New?", NAG Ltd, Oxford, UK.

[21] IBM Research, "IBM Visualization Data Explorer", http://www.research.ibm.com/dx/, IBM, 2000.

[22] "IBM Visualization Data Explorer User's Guide", SC38-0486

[23] "IBM Visualization Data Explorer Programmer's Reference", SC38-0497

[24] Tivoli NetView, http://www.tivoli.com/products/index/netview/, IBM Tivoli, 2000.

[25] Douglas C. Schmidt, "The Distributed Object Visualization Environment", http://www.cs.wustl.edu/~schmidt/dove.html, Department of Computer Science, Washington University, St. Louis, 1999.

[26] Douglas C. Schmidt and ACE-TAO Research group, "Real-time CORBA with TAO", http://www.cs.wustl.edu/~schmidt/TAO.html, Department of Computer Science, Washington Uuniversity, St. Louis, 2000.

[27] Christopher D. Gill, David L. Levine, Carlos O'Ryan and Douglas C. Smith, "Distributed Object Visualization for Sensor-Driven Systems", Proceedings of the 18th IEEE/AIAA Digital Avionics Systems Conference (DASC), St. Louis, Missouri, October 24-29, 1999.

[28] Java Beans, http://java.sun.com/products/javabeans/, Sun Microsystems, Inc, 2000.

[29] Object Management Group, "CORBA Event Service Specification", OMG document number: 00-06-15, 2000.

[30] Sun Microsystems, "Java Beans API Specifications", Edited by: Graham Hamilton, Sun Microsystems, July 1997.

[31] Hans-Erik Eriksson, Magnus Penker, UML Toolkit: John Wiley & Sons, Inc., 1998.

[32] Patrick Naughton and Herbert Schildt, Java 2: The Complete Reference, McGraw Hill Publishing Company, 1999.

[33] Guru Parulkar, Douglas C. Schmidt, Eileen Kraemer, Jon Turner and Anshul Kantawala, "An Architecture for Monitoring, Visualization, and Control and Gigabit Networks", IEEE Network, Volume 11, Number 5, September, 1997.

[34] Douglas C. Schmidt, "Network Monitoring, Visualization, and Control for High-Speed, Large-Scale Networks", http://www.cs.wustl.edu/~schmidt/NMVC.html, Washington University, 2000.

[35] Mackinlay J., "Automating the design of graphical presentations of relational information", ACM Transactions on Graphics, Volume 5, Number 2, Pages 110-141, 1986.

[36] Wehrend S. and Lewis C., "A problem-oriented classification of visualization techniques", Proceedings of IEEE Visualization '90, Pages 139-143, 1990.