Abstract:
The development of distributed applications for scientific or other purposes has always required extensive knowledge not only of the application domain, but also of the underlying technical infrastructure of the computer systems and software that is used to distribute the applications. Pellucid presents a system based on the Common Object Request Broker Architecture (CORBA) [8, 1] and the JavaBeans [6, 7, 3] event model that requires only minimal infrastructure and support knowledge of the scientist that wishes to develop a distributed system. The components can be used with any JavaBeans compatible, free or commercial development environment, and it only employs distributed technology that is available free of charge. All components are highly platform portable and the environment in which they execute does not have to be homogeneous. In this Pellucid presents both a very powerful, but yet simple model for the development of distributed systems, primarily of scientific applications. The potential for Pellucid is demonstrated by sample applications consisting of components for matrix manipulation.

PELLUCID: AN ENVIRONMENT FOR DISTRIBUTED APPLICATIONS

by

Marcus Giese

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

July 2000

N378
G364

APPROVAL

of a thesis submitted by

Marcus Giese

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

Dr. Gary Harkin	[illegible] (Signature)	7/24/00 Date

Approved for the Department of Computer Science

Dr. Denbigh Starkey	[illegible] (Signature)	7/24/00 Date

Approved for the College of Graduate Studies

Dr. Bruce McLeod	Bruce R. McLeod (Signature)	7-24-00 Date

STATEMENT OF PERMISSION TO USE

Signature [illegible]

Date 7/24/2000

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The development of distributed applications for scientific or other purposes has always required extensive knowledge not only of the application domain, but also of the underlying technical infrastructure of the computer systems and software that is used to distribute the applications. Pellucid presents a system based on the Common Object Request Broker Architecture (CORBA) [8, 1] and the JavaBeans [6, 7, 3] event model that requires only minimal infrastructure and support knowledge of the scientist that wishes to develop a distributed system. The components can be used with any JavaBeans compatible, free or commercial development environment, and it only employs distributed technology that is available free of charge. All components are highly platform portable and the environment in which they execute does not have to be homogeneous. In this Pellucid presents both a very powerful, but yet simple model for the development of distributed systems, primarily of scientific applications. The potential for Pellucid is demonstrated by sample applications consisting of components for matrix manipulation.

CHAPTER 1

EARLIER WORK

The development of distributed applications for scientific or other purposes has always required extensive knowledge not only of the application domain, but also of the underlying technical infrastructure of the computer systems and software that is used to distribute the applications. Existing distributed computing environments require the scientist or other potential user to learn a large set of library calls to properly distribute data and synchronize execution. In addition, these environments do not work well with existing Graphical Integrated Development Environments (IDE) such as the Microsoft VisualStudio suite or one of the many Java GUI design environments.

It is nearly impossible to list all prior work and all more or less successful attempts at distributed computing. There are some very successful systems and some that are not very widely used. In the following we briefly examine some of the more well known systems. In addition we describe an earlier attempt by the author to create Pellucid using a Java Jini and JavaSpaces environment as the underlying distributed infrastructure and distribution mechanism.

In general earlier efforts can be categorized into parallel computing environments and distributed environments. For the purpose of this thesis we will focus only on those that fall into the latter group, though some of the parallel environments such as Trapper [18] may offer interesting ideas that could be used to expand Pellucid's functionality. Also not discussed are Web-based parallel management and monitoring tools such as Conspector [19].

The most widely known and used distributed computing environments are Message Passing Interface (MPI) [23] and Parallel Virtual Machine (PVM) [26]. PVM is a distributed computing environment that allows applications to be developed and distributed across a heterogenous compute environment at runtime. PVM consists of a set of libraries that can be used to develop distributed applications in several languages. Several add-ons exist that provide features such as load balancing. Using PVM requires that the application developer have a high degree of programming expertise. To the author's knowledge no graphical design tools exist for use with PVM. There are however graphical PVM tools for debugging and monitoring the distributed application.

MPI is a message passing library that can be used by application developers to create distributed applications. As with PVM many graphical tools exist for the management and monitoring of distributed MPI applications [25], but it also requires a high level of programming expertise.

More recent distributed application environments include JavaParty [15] and Parallel Java [27], but again, the level of programming expertise required is substantial.

Many of the existing efforts implement their own load-balancing and management efforts whereas Pellucid uses an industry-standard environment allowing the use of existing infrastructures and tools. For example some CORBA implementations provide load-balancing and fail-over mechanism that Pellucid can automatically take advantage of. The same is true for the debugging and monitoring of executing applications.

Aside from the non-standard architecture and infrastructure designs, all of these environments assume that the user has a fairly high level of programming experience and understanding of the technical aspects of distributed programming. Pellucid on the other hand offers a true separation of the domain knowledge from the technical and infrastructure knowledge, assuming that the user has a solid understanding of the problem he or she is attempting to solve, but relieving him or her from knowing about the actual distribution and programming details. Using Pellucid to develop requires no programming skills past the ability to use one of the many JavaBeans-enabled Graphical Integrated Development Environments such as Symantec VisualCafe of Inprise JBuilder.

The author developed a version of Pellucid using Sun Jini [11, 10, 28] and JavaSpaces [4] but the reference implementation of Jini provided by Sun Microsystems proved to be too unreliable to be used for a working system. A solution based on Jini and JavaSpaces is attractive because it offers a standard-compliant, Java-only or pure Java option, which also provides a highly fault-tolerant and persistent dataspace with JavaSpaces.

CHAPTER 2

PELLUCID

Pellucid presents a system based on the Common Object Request Broker Architecture and the JavaBeans event model that requires only minimal infrastructure and support knowledge of the scientist that wishes to develop a distributed system. The components can be used with any JavaBeans compatible, free or commercial development environment, and only employs distributed technology that is available free of charge. In this, Pellucid presents both a very powerful, but yet simple model for the development of distributed systems, primarily of scientific applications. The author will demonstrate the potential of Pellucid through the use of matrix manipulation components that are used to create a demonstration application for distributed matrix calculations. It should be emphasized that the examples in this paper show some of the infinite number of components and applications that can be developed based on Pellucid. Pellucid provides a general purpose approach for developing components that inherit the ability to dynamically move around a distributed computing infrastructure.

Distributed systems offer the potential of improved performance, increased fault-tolerance, scalability, and cost savings over large and expensive parallel computers. The main problems with distributed applications has been difficulty of designing and implementing them.

Combining a modern distribution component model architecture (CORBA) and infrastructure with an easy to use and popular complementary event and application model (JavaBeans), writing distributed applications can be simplified. This is made possible by providing often-used operations in a componetized form that is compatible with a de facto object and component standard such as JavaBeans. Every component provided uses inheritance and aggregation to obtain the ability to be distributed and executed on a network of compute nodes. In essence this establishes mobility of fine-grained objects combined with the JavaBeans "click-and-connect" application design model.

In the past Java has sometimes been viewed as slow and not suitable for scientific applications [17, 16], many of which are compute and/or I/O intensive. Advances in the Java compilers and the Java virtual machines over the last year have made the performance issue much less of a problem [24, 20]. The ability to easily distribute applications over a wide array of available compute platforms can make up for the potential shortcomings in the performance of individual members of this compute environment.

Pellucid implements a system of distributed JavaBeans components that allow matrix manipulations. The basic architecture and design allows scalar datatypes, one dimensional arrays, as well as two- and three- dimensional matrices of any valid non-primitive Java datatype. Because of Java's introspection and reflection mechanisms all operations could easily be enhanced to transparently handle any other datatype.

The arithmetic operations are built on top of a CORBA infrastructure that allows every component to move itself to any network computer that is running a Pellucid server or that is configured to allow an automatic activation of a Pellucid server. The server is not specific to any particular arithmetic function or component, but rather serves as a container to execute any component that follows the Pellucid interface specifications. In addition to the actual distribution infrastructure, Pellucid components are complemented by a mechanism that allows the connection of CORBA events to JavaBeans events, allowing a seamless integration of the JavaBeans UI design with the remote CORBA events. This allows a application developer to visually design the application in a JavaBeans UI environment, selecting JavaBeans components and visually connecting the components. All components are provided with the name of a default machine to execute on, but a JavaBean [1] Property Editor for every JavaBean allows the application designer to visually designate components for execution on specific remote machines.

The following example demonstrates Pellucid's basic underlying principle. The matrix expression:

$$result = ((A + B) + (B * C)) * ((B * C) * (C - D)) \tag{2.1}$$

can be described with the following task graph, shown in Figure 1.

Every node in the task graph is implemented by Pellucid using a corresponding JavaBean object and each object can execute independently on a different machine in the network.

Pellucid operations use Java Introspection and Reflection combined with polymorphism to support the implemented operations for many different data types. The

[1] A JavaBean is a Java class that follows certain naming conventions and patterns

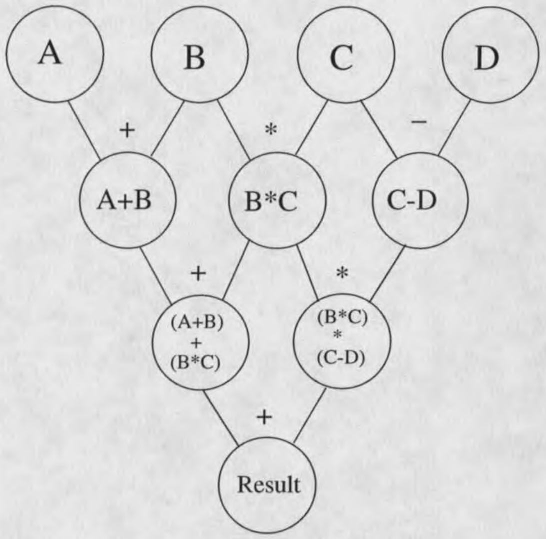


Figure 1. Task Graph of simple arithmetic.

operations supported by the example applications used to demonstrate the Pellucid approach and architecture are:

- Matrix Multiplication
- Matrix Addition
- Matrix Subtraction
- Matrix Inversion
- Identity Matrix
- Input
- Output

All operations are built on top of Pellucid infrastructure support functions that provide communication abilities, fail-over support, execution monitoring and load and node management. Pellucid implements some of these services directly while others are leveraged from the CORBA infrastructure. Support for communication and transfer of executable code is implemented by Pellucid components, using low-level CORBA marshalling functions and interfaces. Event notification and mapping is implemented by Pellucid using the CORBA call-back mechanisms and load-balancing and fail-over support is only available to the extent supported by the underlying CORBA ORB implementation that is used to deploy Pellucid-based applications, and will differ between CORBA implementations.

Pellucid is unique in its use of the JavaBeans event model to enforce proper operation sequencing and synchronization. As will be briefly explained below, no node in the task graph of shown in Figure 1 can procede with its execution until its input has been received from preceding operations. This ensures proper sequencing of operations without requiring the use of complex and cumbersome synchronization calls. Pellucid extends this single-machine event model to a distributed environment.

CHAPTER 3

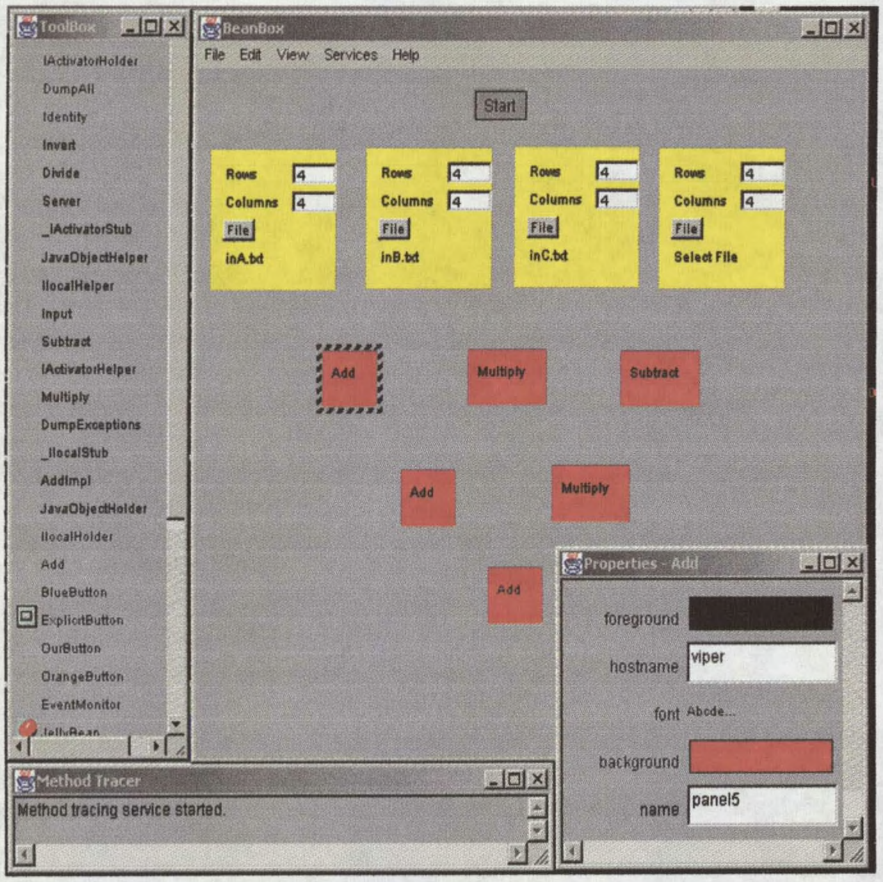
METHODS

User Interface

When selecting a potential User Interface (UI) architecture many possibilities exist. Simplicity and ultimate portability are desirable goals for Pellucid for that reason Java, and more specifically JavaBeans, was chosen for the User Interface implementation of the components. All UI portions of the arithmetic operations were implemented as visual JavaBeans painted in a basic AWT Applet [21]. Figure 2 shows a simple matrix manipulation application created using Sun Microsystems' BeanBox that is distributed with the JavaBeans development kit [29].

Figure 3 shows the same application after generating and running the application as a Java Applet. The applet was generated by the BeanBox using only default values. This is just one of the unlimited number of applications that can be created using the Pellucid distributed architecture. The example shown implements the linear equation shown in Figure 1.

In the left part of Figure 2 the various components that are available for creating an application are shown. Simply selecting a component from the pallet and moving it to the design canvas adds that component to the application. Components can be connected through the Edit menu of the BeanBox which allows the application designer to select from the list of operations a component supports and connect these operations to those of other components on the canvas. The Input and arithmetic



Rows 4, Columns 4

Method tracing service started.

foreground

hostname viper

font Abcde...

background

name panel5

Figure 2. Sun BeanBox.

components are Pellucid specific, and the Start button is a standard component available in the BeanBox. Components such as the Input bean have additional controls such as file selection box and fields for specifying the size of the input matrices.

The lower right corner of Figure 2 shows the Property Editor for the selected Add component. Visible is the computer name that the component should execute on (viper). The machine name can be changed just as easily as any of the other properties, the color of the component for example.