NASA/CR-2000-210545
ICASE Report No. 2000-39

Arcade: A Web-Java Based Framework for Distributed Computing

Zhikai Chen and Kurt Maly
Old Dominion University, Norfolk, Virginia

Piyush Mehrotra
ICASE, Hampton, Virginia

Mohammad Zubair
Old Dominion University, Norfolk, Virginia

ICASE
NASA Langley Research Center
Hampton, Virginia
Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

October 2000

ARCADE: A WEB-JAVA BASED FRAMEWORK FOR DISTRIBUTED COMPUTING*

ZHIKAI CHEN†, KURT MALY†, PIYUSH MEHROTRA‡, AND MOHAMMAD ZUBAIR†

Abstract. Distributed heterogeneous environments are being increasingly used to execute a variety of large size simulations and computational problems. We are developing Arcade, a web-based environment to design, execute, monitor, and control distributed applications. These targeted applications consist of independent heterogeneous modules which can be executed on a distributed heterogeneous environment. In this paper we describe the overall design of the system and discuss the prototype implementation of the core functionalities required to support such a framework.

Key words. meta-computing environment, multi-module applications

Subject classification. Computer Science

1. Introduction. Distributed heterogeneous environments are being increasingly used to execute a variety of large size simulation and computational problems. For instance, in multidisciplinary optimization, multiple heterogeneous modules interact with each other to solve an overall design problem. Typically these modules, consisting of various C or Fortran programs, are developed as separate codes, e.g., structural or flow analysis of an aircraft configuration, and are optimized independently. The traditional path for integrating these modules, through the use of scripts makes the process of specifying and optimizing the overall design of such applications, a long and tedious process often taking several weeks. The slowness of this process is mainly due to the absence of a collaborative environment where (i) different modules and their interactions can be specified, and where (ii) testing, monitoring, and steering of the overall design can be done by multiple users from different disciplines concurrently. In this paper we describe Arcade, a web-based environment for designing, executing, monitoring, and controlling distributed heterogeneous applications.

A typical scenario for developing and executing a distributed application is as follows: a team of designers collaboratively develops the application consisting of a hierarchical set of modules. That is, individual members are made responsible for specifying the submodules while the project leader is responsible for the overall integration of the application, i.e., connecting the outputs of one module to the inputs of another. The modules can range from simple sequential programs, to data-parallel programs capable of execution on a multiprocessor or a network of workstations, to more complex subsystems which are defined hierarchically through the use of submodules. Preexisting modules whose sources are not available may need to be "wrapped" in order to plug them into the overall application.

Once developed, the application is executed in a distributed environment using a heterogeneous network of workstations and multiprocessor machines. During the execution, team members sitting at their individual workstations simultaneously monitor the flow of progress of the application. That is, the team members can see the currently executing modules at any level of the hierarchy. They can also view the intermediate data flowing between different modules including large data sets using visualization tools.

A team member responsible for a particular subsystem can change data values under the control of the subsystem in order to steer the computation in the right direction. The team member can also dynamically

*This work was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23681.

†Computer Science Department, Old Dominion University, Norfolk, VA 23529 ({chen_z, maly, zubair}@cs.odu.edu).

‡ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23681-2199 (email: pm@icase.edu).

alter the control flow if necessary. For example, in a design cycle, the responsible team member may decide that a particular module is not affecting the optimization and may bypass the module by using old values in each cycle. Similarly, the team could decide to use another algorithm and replace a module with another plug compatible module. Once the execution is complete, team members again examine the final results using the visualization tools.

The overall goal is to design an environment which is easy to use, easily accessible, portable and provides support through all phases of the application development and execution. We plan to leverage off of commodity technologies, such as the Web and Java, to implement various parts of the environment. These technologies are capable of seamlessly interconnecting disparate hardware platforms running different operating systems across diverse locations providing an ideal environment for distributed simulation of complex systems.

In this paper we first describe our overall goals and then discuss the current prototype reporting on our experiences and problems in implementing the system. The rest of the paper is organized as follows. In the next section we present some related work. The two following sections describe the overall Arcade architecture and the current prototype, respectively. The last section focuses on future work and conclusions.

2. Related Work. Several software systems have been developed that make distributed computing available to an application programmer. These can be distinguished into different categories. The first category of environments includes systems such as MPI [10], PVM [15], pPVM [9] and JAVADC [5]. All these environments support distributed computing in varying degrees of generality; however, either they are not web based or they lack collaborative features. Also, they are mostly suitable for running SPMD programs. The second category of environments address large distributed heterogeneous codes but are focused on a single application domain. Examples of such environments include FIDO [16] and MIDAS [13]. However, both these systems are either hardwired to a specific problem area or are too restrictive. The other major limitation is that they lack a collaborative environment which would permit different members in a group to interact with the application at various stages of its design and execution.

The third category of environments which includes IceT [7], Programmer's Playground [6], PRE [14], and WebFlow [3] supports some forms of heterogeneous distributed applications. The front-end for most of these systems, is generally some variation of large-grained data flow graphs with modules being triggered when their inputs are available. In our experience, we have found that to more control structure than provided by such data-flow based systems is required to easily express heterogeneous applications. For example, in a multidisciplinary optimization code, the optimization cycle would have to be embedded within a module in a system which only supports data-flow rather than being explicit at the outer level. Also, these systems mainly concentrate on different aspects of the infrastructure required for managing the execution and interaction of the modules making up the application whereas the goal of the project described here is to build an integrated framework for all phases of the design and execution of distributed heterogeneous applications. Note that systems such as Tango [1] and Habanero [11] focus on interactive collaboration between users. Such technologies would be useful in the specification and the monitoring phases of the framework being proposed here. However, such collaborative systems do not provide any support for the management and steering of the execution of distributed applications.

3. Arcade Architecture. The architecture of the proposed framework is divided into three tiers as shown in Figure 1.

First Tier: The first tier consists of the Java applets providing the following interfaces to the users:

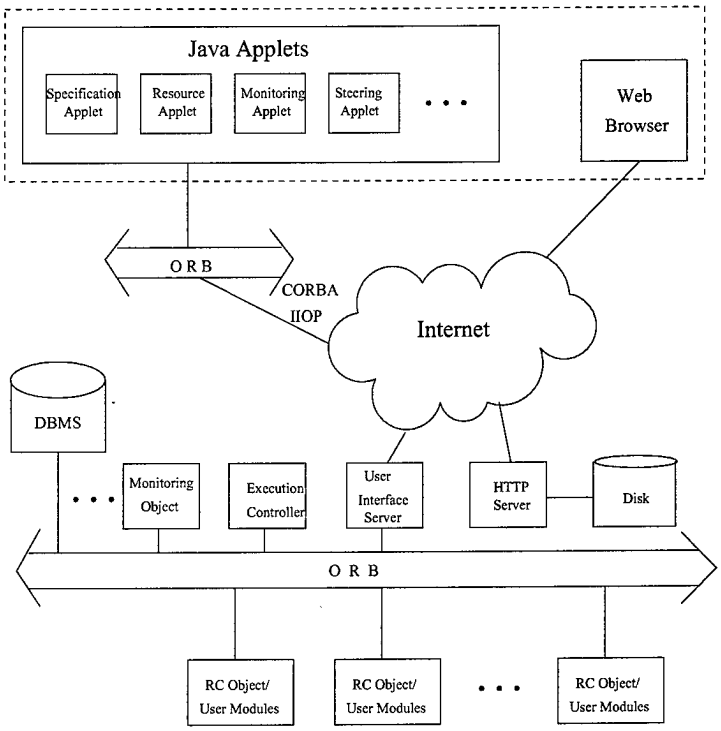


FIG. 1. *Web Object Infrastructure for the Proposed Framework*

- *Application design interface* for the hierarchical specification of execution modules and their dependencies. The system will provide support for multi-user specification of hierarchical modules including the specification of module interactions and database access for persistent storage of results.
- *Resource allocation and execution interface* for specifying the hardware resources required for the execution of the application. The resources could be chosen by the users or by the system based on the current and predicted loads of the system and the characteristics of the application. The choice could be made statically or dynamically during the execution of the application. The system will also allow the users to choose the input/output files and any command line arguments for the modules prior to starting the execution.
- *Monitoring and steering interface* for monitoring and controlling the execution of the application. Multiple users will be allowed to monitor the run both the flow of execution and the intermediate results. However only the subteam responsible for a particular submodule will be allowed to steer its execution by either modifying data values or replacing plug-compatible modules.

Middle Tier: The middle tier consists of logic to process the user input and to interact with application modules running on a heterogeneous set of machines. The overall design is a client-server based architecture in which the *Interface Server* interacts with the front-end client to provide the information and services as needed by the client. When the user requests the execution of an application, the Interface Server launches an *Execution Controller (EC)* which manages the overall execution of the application by firing up user modules on the specified resources as and when required. Other objects in the middle tier handle any monitoring and steering requests from the client.

Third Tier: The third tier consists of *Resource Controllers (RC)* and the *User Application Modules.* Each active resource in the execution environment is managed by an RC which is responsible for launching modules on the resource and also for interacting with the Execution Controller in order to keep track of the executing applications.

The main advantage of a three-tier system is that the client or the front-end becomes very thin, thus making it feasible to run on low-end machines. Also, since most of the logic is embodied in the middle tier, the RCs can be kept lightweight thus keeping the additional loads on the executing machines to a minimum.

The Web Object approach we are taking will coexist with the regular HTTP server. In contrast to approaches with CGI-to-CORBA gateway or HTTP-to-IIOP gateways, this approach is easier to implement [12]. Here, the HTTP server will provide users Web pages with Java applets. These applets will interact with the CORBA Interface server using CORBA-IIOP protocol. The applets can use static IDL-generated client stubs or a Dynamic Invocation Interface to interact with the CORBA Interface server. The CORBA Interface server will interact in similar manner with other objects on the ORB bus (see Figure 1).

4. Arcade Prototype. We have implemented the three-tier system, as described in the last section, in a prototype Arcade system. The current system allows single users to specify applications through either an offline script-based system or through a visual interface. The resources required for the execution have to be statically specified by the user. The current prototype supports only file-based interaction among the modules. The system manages the execution of the modules on a network of workstations in a single domain. The execution status of the application can be monitored by multiple users simultaneously.

4.1. Application Specification. In our framework a distributed application consists of a collection of heterogeneous modules (application codes from different disciplines). We are targeting applications where these modules are very coarse-grained. A typical distributed application requires that these modules be executed in a specific order and possibly on different machines. For certain problems a set of modules may need to be executed iteratively until a desired optimization criteria is reached. To be able to support a wide variety of distributed applications, we support the following types of modules:

- *Normal Module*: This is the basic module in our framework and is used to represent the executable parts in the applications. A normal module is identified by its executable code and its input/output file requirements.
- *If Module*: This module provides a mechanism for testing the value of a condition. The truth-value of the condition determines whether the modules in the then-block or the else-block (if present) will be executed.
- *Loop Modules*: These modules allow a set of "internal" modules to be iteratively executed. There are three kinds of looping modules: *For* module for a predetermined module of iterations, *While* module: where the iteration condition is tested at the beginning of the loop, and *Repeat* module in which the condition is tested at the end.
- *SPMD Module*: A module representing a SPMD program written using one of the message passing interface like MPI, PVM, etc. This module, depending on its specification, gets executed on a dedicated parallel machine or on a cluster of workstations.
- *Hierarchical Module*: An abstract module representing a subgraph, i.e., a recursively defined collection of modules.

In the current prototype there are two ways to specify a distributed applications: *script-based* or *visual.* In the next two subsections we describe these two specification mechanisms.

Script-Based Specification. Arcade supports script-based offline specification of a distributed heterogeneous application. The syntax of the script is simple allowing users to specify the different modules, as described above along with the properties of these modules. In particular, the user must specify the following properties for each *Normal Module*:

- Module Name
- Module Directory: the directory in which the executable and input/output files are to be found.
- File Name: the name of the executable.
- Parameter: command line arguments to be used for execution.
- Machine Name: on which the module is to be executed.
- Input Names and Files: a globally unique name for each input along with the associated file name.
- Output Names and Files: a globally unique name for each output along with the associated file name.

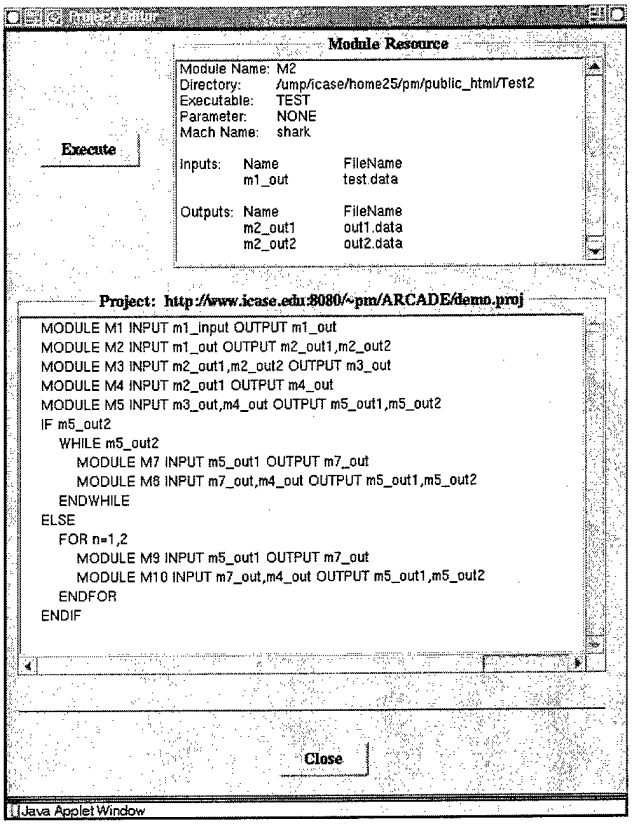
Similarly, the user can specify the properties of the other types of modules.

Figure 2 shows a screenshot of the script-based specification of an artificial application whose structure is shown graphically at the bottom. In the script-based system, the interconnections between the modules are defined based on the globally unique names for the inputs/outputs of the modules. Thus, for example as shown in the top half of Figure 2, Module *M2* has two outputs: *m2_out1* and *m2_out2* while Module *M3* has two inputs: *m2_out1* and *m2_out2* and Module *M4* has one input: *m2_out1* defining the interconnections shown graphically in the bottom half of the figure.

Visual Interface. The visual specification applet allows a user to graphically specify a heterogeneous application. The objective is to support a visual specification which is: (i) intuitive to build, (ii) can be used for visual monitoring, and (iii) works with the Web. There exist a number of visual language projects – see [4] for a classification of many of these projects. However, most of the projects which support program specification are either focussed on fine-grained programming or support only data-flow applications. That is, they do not provide any integrated, intuitive approach to specify control constructs in coarse-grained distributed applications.

We have implemented a Java applet that provides a visual specification interface and addresses some of these issues (see Figure 3). The visual specification can be seen as a graph where a node represents a module and the arcs represent the flow of data between the modules. It is easy to see how a data flow-based application can be modeled using such a system. It becomes a little trickier to accommodate control structures such as conditionals and iterations, in particular when we want to use the visual specification for monitoring too. We accommodate if-modules and loop-modules by restricting their bodies to be hierarchical modules which are specified through a separate window. Thus, the modules labeled *Then-block* and *Else-block* represent hierarchical modules abstracting the then and else part of if construct respectively. Similarly, the module *Body*, represents the loop body of the while loop. Restricting the bodies of control structures to hierarchical modules eases the task of specification and allows the application to be visually represented. However, it does not provide an integrated view of the whole application in a single window, i.e., the body of a control structure is always shown in a separate window.

4.2. Application Execution. Each application is internally represented by a Java `Project` object. The `Project` object, consisting of a vector of modules objects, is the central object in our framework. All the information related to the application, both static and dynamic, is stored within this object. The `Project` object is a complex object that is shared by all the processes of the middle tier (see Figure 1) and supports methods that are used by these processes. When the user requests the execution of an application, the *User*



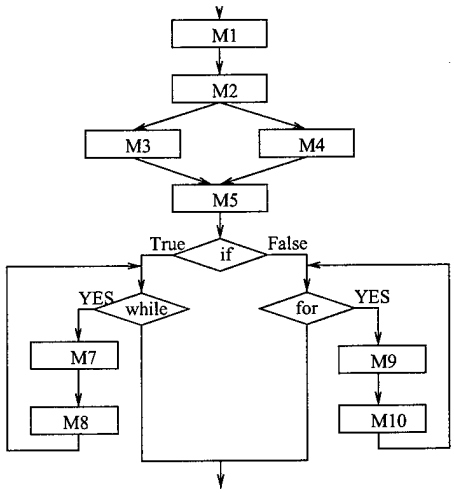


FIG. 2. *Script-based Specification*

Interface Server passes the corresponding `Project` object to the *Execution Controller (EC)*. It is the EC's responsibility to manage the execution and the interaction of the modules specified within the application.

For executing the application, the EC needs to call some initialization methods of the `Project` object followed by its execution method. For example, if the application has been specified using the script-based mechanism, then the dependencies (which had been specified implicitly) have to be explicitly computed and

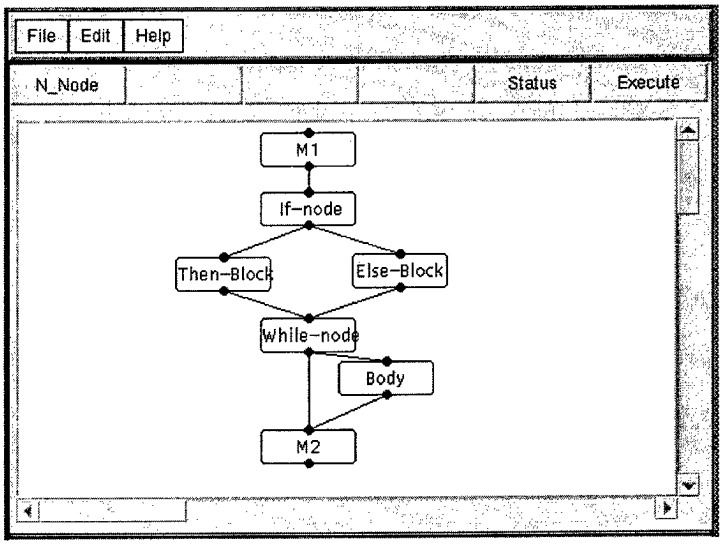


FIG. 3. *Visual Specification Interface*

stored. The EC then executes the modules using a data-flow approach. That is, it launches the execution of a module as soon as its inputs are available. Note that the interaction in the current system is based on files. Thus, the EC has to ensure that the inputs for a module are in the files as specified in the project specification. That is, if a module produces a file which is physically different from the file specified as an input file by a dependent module the EC copies the file over before execution is started. Here we are assuming that the application is executing in a single domain with a global file system so that copying files does not raise any security issues.

The system requires the user to specify the resource on which each module is to be executed in the application specification. Thus, to execute a module, the EC contacts the Resource Controller (RC) on the specified resource and requests that the module be executed. The RC starts the execution and monitors it. The RC notifies the EC when the module finishes execution. The EC determines the modules that were dependent on this module and launches them if all of their inputs are ready. Once all the modules have finished execution, the user can be notified.

A monitoring applet allows users to monitor the execution status of the application. Two different interfaces supported: text-based and graphical. The text-based interface indicates the time a module execution begins and the machine being used for the execution. As a module finishes execution, the completion time is also indicated. The graphical interface is available only for visually specified applications and uses a pre-determined color-scheme to indicate modules which have finished execution, are currently executing and are awaiting execution.

5. Future Work. In the last two sections we have described our overall design of the Arcade system and the state of the current prototype. This is work in progress and we are working to improve and extend several aspects of the system. We discuss some of these issues here.

The specification interface allows both a script-based and visual specification of the application. However, both of these interfaces provide only restricted support for specification. We are experimenting with several ways to extend the specification script, in particular examining how we can use a full programming language such as Java as the specification mechanism. Also, the current interface only displays scripts which have

been edited offline - we are building a script editor so that the application can be specified from within the framework. The visual interface will also be extended to make it more robust and integrated. Currently there are no multi-user specification capabilities. We are extending the system to include the concept of teams including managers and subteams and support collaborative specification of applications.

To make our framework easy to use, we would like it to provide support for massaging input and output from different modules. This is sometimes necessary to make the output of a module compatible with the input of the other modules. One way around the problem is to require the module designer to make the necessary changes in the code. However, we are examining mechanisms which will allow "data translators" to be specified at a high level and automatically generated so that data can be transformed before they are communicated from one module to another. We are exploring two different ways of doing this. First, we are investigating the use of SmartFiles [8], a system which supports the specification of meta-data to define data contained in traditional files. Second, we are also investigating a tool which provides similar facilities, however, this tool uses a data base as an intermediary for storing and transforming the data.

Another issue is resource mapping. In the current system, we require the user to statically specify the resource on which each module is to execute. Our hope is to include an intelligent resource mapper which can dynamically determine the optimal mapping of a module given its requirements and the current loads of the system. We are investigating several options, including incorporating a system such as AppLeS [2] into Arcade.

The two major components of security that are of interest to us are: authentication, and access control. These issues are manageable if we restrict our framework to one domain. However, it is not clear how to address these issues when working across organizations. The main reason is the existence of multiple non-inter-operable standards that are transport-dependent. In the current prototype we have not addressed the multi-domain security issues.

Another major challenge, besides looking at the specific interfaces and controllers, is to architect the framework such that it is possible to plug-and-play different interfaces and controllers. We are hoping that the CORBA-based approach that we are taking for the future extensions to the framework, will make this task easier.

6. Conclusion. In this paper, we have described a integrated Web-Java based environment, Arcade, for the design, execution, monitoring and steering of heterogeneous applications in distributed execution environment. We have described the current prototype of the system which is capable of executing distributed applications on a network of resources in a single domain. We are currently expanding the system to incorporate all the facilities envisioned in our architecture in a phased approach, an approach where the driving force is the user of the system. More information on the Arcade system can be found at http://www.icase.edu/arcade.

REFERENCES

[1] Lukasz Beca, Gang Cheng, Geoffrey C. Fox, Tomasz Jurga, Konrad Olszewski, Marek Podgorny, Piotr Sokolowski, Tomasz Stachowiak, and Krzysztof Walczak, *TANGO — A Collaborative Environment for the World-Wide Web*, http://www.npac.syr.edu/projects/tango.

[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, *Application-level Scheduling on Distributed Heterogeneous Networks*, Supercomputing'96, November 1996.

[3] D. BHATIA, V. BURZEVSKI, M. CAMUSEVA, G. FOX, W. FURMANSKI, AND G. PREMCHANDRAN, *WebFlow — A visual programming paradigm for Web/Java based coarse grain distributed computing*, Concurrency: Practice and Experience, Java Special Issue, 9(6) (March 1997), pp. 555-578.

[4] M. BURNETT AND M. BAKER, *A Classification System for Visual Programming Languages*, Technical Report 93-60-14, Department of Computer Science, Oregon State University, Corvallis, OR 97331, 1993 (revised 1994).

[5] Z. CHEN, K. MALY, P. MEHROTRA, P. VANGALA, AND M. ZUBAIR, *Web-based Framework for Distributed Computing*, Concurrency: Practice and Experience, Java Special Issue, 9(11) (November 1997), pp. 1175-1180.

[6] K.J. GOLDMAN, B. SWAMINATHAN, T.P. MCCARTNEY, M.D. ANDERSON, AND R. SETHURAMAN, *The Programmers' Playground: I/O Abstraction for User-configurable Distributed Applications*, IEEE Transactions on Software Engineering, 21(9) (September 1995), pp. 735-746.

[7] P. GRAY AND V. SUNDERAM, *IceT: Distributed Computing and Java*, Concurrency: Practice and Experience, Java Special Issue, 9(11) (November 1997), pp. 1161-1168.

[8] M. HAINES, P. MEHROTRA, AND J. VAN ROSENDALE, *SmartFiles: An OO approach to data file interoperability*, in Proceedings of OOPSLA 95, the Tenth ACM Conference on Object-oriented Programming Systems, Languages, and Applications, Austin, TX, pp. 453-466, October 1995.

[9] K. MALY, S. KELKAR, AND M. ZUBAIR, *Scientific Computing Using pPVM*, International Conference on Parallel Processing, 2 (August 1994), pp. 201-205.

[10] *Message Passing Interface Forum*, MPI: A Message-Passing Interface Standard Version 2.0, Technical Report, Computer Science Department, University of Tennessee, Knoxville, TN, 1997.

[11] NCSA Habanero Project, http://www.ncsa.uiuc.edu/SDG/Software/Habanero/.

[12] R. ORFALI AND D. HARKEY, *Client/Server Programming with Java and CORBA*, John Wiley & Sons, 1997.

[13] J.C. PETERSON, *Multidisciplinary Integrated Design Assistant For Spacecraft (MIDAS)*, http://mishkin.jpl.nasa.gov/Midas_Page.

[14] *Product Realization Environment*, http://www-collab.ca.sandia.gov/pre.

[15] V. SUNDERAM, *PVM: A Framework for Parallel Distributed Computing*, Concurrency: Practice and Experience, 2(4) (December 1990).

[16] R.P. WESTON, J.C. TOWNSEND, T.M. EIDSON, AND R.L. GATES, *A Distributed Computing Environment for Multidisciplinary Design*, 5th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Panama City, FL, AIAA 94-4372, September 7-9, 1994.