

# Using the Grid to Support Software Component Systems \*

Randall Bramley    Dennis Gannon    Juan Villacis    Andrew Whitaker†

## Abstract

The Grid provides a sophisticated service layer allowing users to access and manage distributed hardware resources. Grid tools such as Globus handle multiple networks, machines, binary representations of data, and scheduling policies. However, providing end users with practical tools to tie together combined hardware and software resources distributed across the Grid requires an additional software layer. The Component Architecture Toolkit (CAT) is a system based on distributed software components, which are found, instantiated, run, and connected together through a high-level visual-programming model. This paper describes the architecture underlying the CAT, and shows example computations carried out using it.

## 1 Introduction

The Component Architecture Toolkit comprises subsystems which can be combined to form a high-performance problem-solving environment for computational science and engineering (CS&E). The CAT design emphasizes flexibility and extensibility, so that the full range of CS&E activities can be accommodated. The user-level abstraction that the CAT presents is a component system. In this context, **components** are encapsulated blocks for creating software applications. As illustrated by the applications in section 3, components can range from small computational actions such as scaling an array up to complete parallel finite element analyses and 3D immersive visualization systems. Components can also represent a database server or a remote instrument dynamically providing data under real-time control. Instead of linking and compiling source code, a CAT user modifies and connects binaries in a visual programming environment. Those binaries interact as peers using *ports*, well-defined interfaces specifying input and outputs as well as any “control signals” from the outside. Components can be connected, disconnected, and hot-wired into an application running across globally distributed supercomputers and workstations.

An important contribution of the CAT is to provide a generic problem-solving environment that can be tailored to the needs of an end user. The standard Grid computing model is shown in Figure 1(a), with application codes (in C/C++, Fortran, etc.) at the top, high performance computing (HPC) libraries such as SCALAPACK next, infrastructure services such as Globus, Nexus or network protocols following, and hardware such as supercomputers and visualization equipment at the bottom. Although this is a natural mental model for computer scientists, a better model for end users is in Figure 1(b). Here the bottom layer comprises both hardware *and* software resources: computers, code, visualization engines, databases, and instruments. The next layer provides resource discovery, location and management. The following layer consists of composition tools for

---

\*Work supported by NSF grants CDA-9601632, CCR-9527130, and ASC-9502292

†Department of Computer Science, Indian University, Bloomington, IN

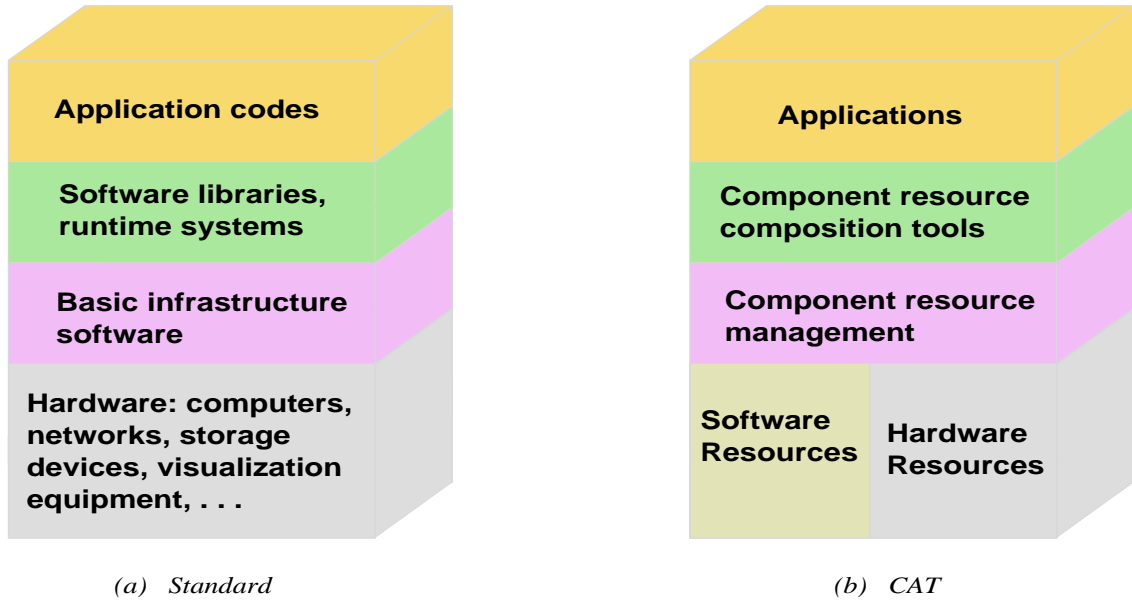


FIG. 1. *Grid computing models*

organizing those resources: visual programming GUIs, scripting languages, or standard programming languages with systems calls. At the top for a user is the application - which does not consist of code, but of the physical processes and mathematical models of interest. Our component model is natural for Grid computing, because the representation of data and even the computer language used to write a component are strictly internal to a component. A user can be shielded from the details of architecture or even location of the components. Equally important, the mental model of connecting components via ports has a natural mapping to the physical model of distributed resources being connected via networks.

## 2 The CAT System

At its core, the CAT is essentially a component container framework that builds upon the notion of *interface programming*. A component's **interface** describes the set of methods that may be invoked on the component. The CAT system is composed of several component subsystems, each of which interacts with other subsystems via well-defined interfaces. This approach allows the CAT to be adapted to whichever subsystem implementation is available to it. A brief overview of the CAT framework follows.

### 2.1 CAT Framework

A diagram of the CAT and its relation to the outside world is shown in Figure 2. In the center is the CAT (on Host D), and connected to it are a set of computational resources (on Hosts B, C, and D). The computational resources are a set of component programs that may run on a variety of machines and in a variety of configurations/environments. The resource information service (**RIS**) contains information relevant to the operation of the computational resources. This may include authentication, location of component binaries, restrictions on component usage, and so forth. Each compute host must also run a special process creation server, called the **proess creator (procreator)**, which spawns processes

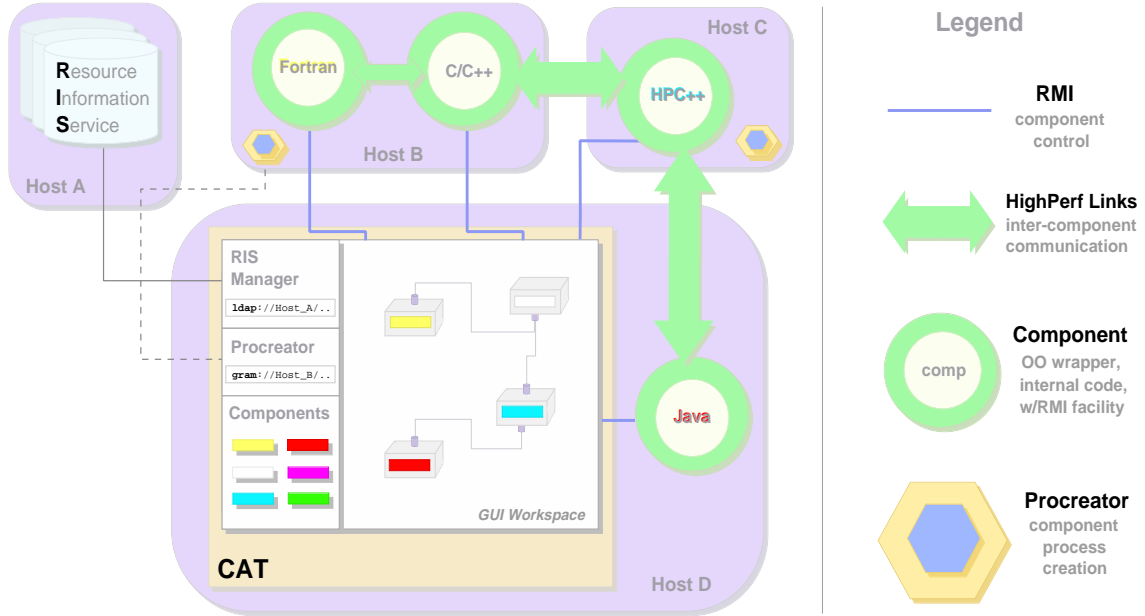


FIG. 2. Overview of the CAT framework

on behalf of the CAT. The procreator encapsulates all of the policies, usage restrictions, job scheduling, etc., that are involved in starting up processes on host machines. Once started, components communicate with each other through high-performance communication links. Components are controlled by users of the CAT through a remote method invocation (RMI) mechanism. In short, the CAT acts as a front-end to the end user by providing a consistent interface to the complex, heterogeneous systems underlying it.

## 2.2 CAT Component Model

In the CAT framework, a **port** is a typed communication channel through which components can delegate external I/O. Each component is furnished with two types of port subcomponents: input and output ports. Output ports may communicate directly with any number of input ports, but an input port may be connected to at most one output port. Inter-component communication is achieved by connecting a component's output port to another component's input port, and passing data between them. Data transfer is initiated (indirectly by the component) via a method call on an output port. In contrast, the CAT communicates directly with components and their ports via RMI.

In general, a component is known to other components only through its public interface. Within the CAT framework, a component must support a base *framework* interface in addition to any other interfaces it may have. The component's framework interface is partitioned into several groups: CAT  $\leftrightarrow$  Component, CAT  $\leftrightarrow$  ComponentInputs, CAT  $\leftrightarrow$  ComponentOutputs, and ComponentOutputs  $\leftrightarrow$  ComponentInputs. Each group defines the scope of interaction between constituent components. Membership in the CAT framework entails that a component contain and implement the framework interface.

## 2.3 CAT Framework Implementation

The choice of how to implement the CAT framework was constrained by several requirements. The intent of these requirements was to insure that the CAT provide an effective

and useful framework in which to build and deploy distributed applications for large-scale scientific computing problems. Chief among these constraints were,

- high-performance communication links; for large-scale scientific problems, the amount of data that may need to be passed between components over the network could range from MB to GB; thus, it was imperative to have a low-latency, high-bandwidth communication protocol/library linking the components
- simplified delineation of work boundaries; users of the system should not have to deal with creating framework tools, but rather should focus on programming component content; however, access to lower-level support tools should be allowed for the purposes of enhancements (to the existing toolset), and performance considerations; hence, programming tools/paradigms/environments that support such delineation would be essential for the adoption, extension, and maintenance of the CAT framework
- facilities to exploit coarse-grained parallelism at the application and component level, as well as explicit support for intrinsically parallel components; the CAT framework should support multithreading at the component level, and multiprocessing activities at the inter-component level
- small resource footprint; the CAT framework should impose relatively low overhead (e.g., minimal use of CPU cycles, memory, network traffic, etc.) in comparison to the resource requirements of the embedded component operations
- use of “off-the-shelf” technology; whenever possible and wherever appropriate, standard open technologies should be used to build the CAT framework

A careful selection of lower-level infrastructure layers that best met the above requirements was made prior to building the CAT. A brief summary of the reasons for our choices are described below.

**2.3.1 Choice of Grid** Globus [10] was chosen to serve as the lowest layer in the Grid computing environment for several reasons. First, it was the most developed in terms of high-performance communication protocol support (Nexus), authentication and component creation schemes (GSS, GRAM), and resource location mechanisms (MDS). Also, it had multi-language support (C/C++, Java) which greatly facilitated higher-level tool building (NexusRMI). Each of these characteristics made Globus the ideal choice for the Grid layer.

**2.3.2 Choice of middleware** Java Beans [8] was selected as the primary middleware platform in which to develop the CAT framework tools. In particular, NexusRMI [9], a Java-based library that allows Java objects to perform RMI calls on (possibly) non-Java objects via the Nexus protocol (and vice-versa), was selected as the mechanism for component control. Also, the CAT graphical user interfaces (see section 3) were written using the JDK1.1 AWT with Swing. The main benefits of Java are its cross-platform support, networking capabilities, and sophisticated object-oriented libraries. Also, since Java is quickly becoming the “de-facto” standard for web programming, this may aid in widespread use of the CAT tools.

**2.3.3 Choice of component language** In the CAT framework, components are constructed in layers. The outermost layer is called the **component wrapper**, and it provides basic framework services such as port management, signal handling, exception handling, etc. The inner layers contain application-specific code which typically performs some resource intensive computation. In order to facilitate the rapid construction of framework components, object-oriented languages such as HPC++ [11], Java, and Fortran90 were chosen as the languages in which to write the component wrappers. The component core, however, could be written in whichever language the wrapper code had bindings to (e.g., C/C++, Fortran, Perl, Python). This flexibility facilitates the incorporation of a wide range of existing “legacy” code into the CAT framework.

**2.3.4 Choice of resource information service** The resource information service (RIS) is implemented in two parts: a distributed directory service, together with a network of CAT registries. A brief description of each is presented below.

The distributed **directory service** contains a hierarchy or tree of data entries that may be spread across multiple directory servers. Typically, a directory service is optimized for reading more than writing, and is used to store mostly static information.

The RIS uses a directory service based on the Lightweight Directory Access Protocol (LDAP). LDAP offers several advantages over conventional databases for the purpose of constructing an information service for a component toolkit. First, LDAP is an emerging standard that is gaining industry-wide support from major software manufactures such as IBM, Sun, Netscape, and Novell. Second, LDAP can store any form of data that can be serialized into a binary format. For example, component meta-data (e.g., port type information in the form of a Java class file) or even the component binary itself can be stored directly in the LDAP database. Third, LDAP entries can contain “pointers” to other entries, thereby reducing the amount of replication in the database. In particular, the RIS extends the Globus Metadata Directory Service (MDS) (which uses LDAP as its underlying database) by adding component software entries cross-referenced with the hardware-specific MDS database entries. Finally, LDAP can be configured to use strong security mechanisms to restrict access to specific parts of the database.

The second part of the RIS is the network of CAT registries. A CAT **registry** is a simple, flat database containing named references to live instances of components, and is implemented on top of the Java RMI registry. Component instances which are created during a CAT session (described in section 3.2) are automatically added to the CAT’s registry. The user of the CAT may “publish” the location and contents of the registry to the LDAP server so that other users of the RIS may use those components (subject to user and/or host access controls). Also, since a CAT registry can be run independently of a CAT session, active components can continue to be accessible through an attach/detach mechanism long after the original (parent) CAT session has shut down.

### 3 CAT Applications

In this section, we present an overview of the how a user interacts with the CAT, and describe several functions and features available through the CAT. Later, we provide examples of how the CAT has been used in real CS&E activities.

#### 3.1 CAT Session

A user interacts with the CAT through a graphical workspace manager. The workspace manager operates in tandem with an InfoBrowser, a graphical front-end to the RIS.

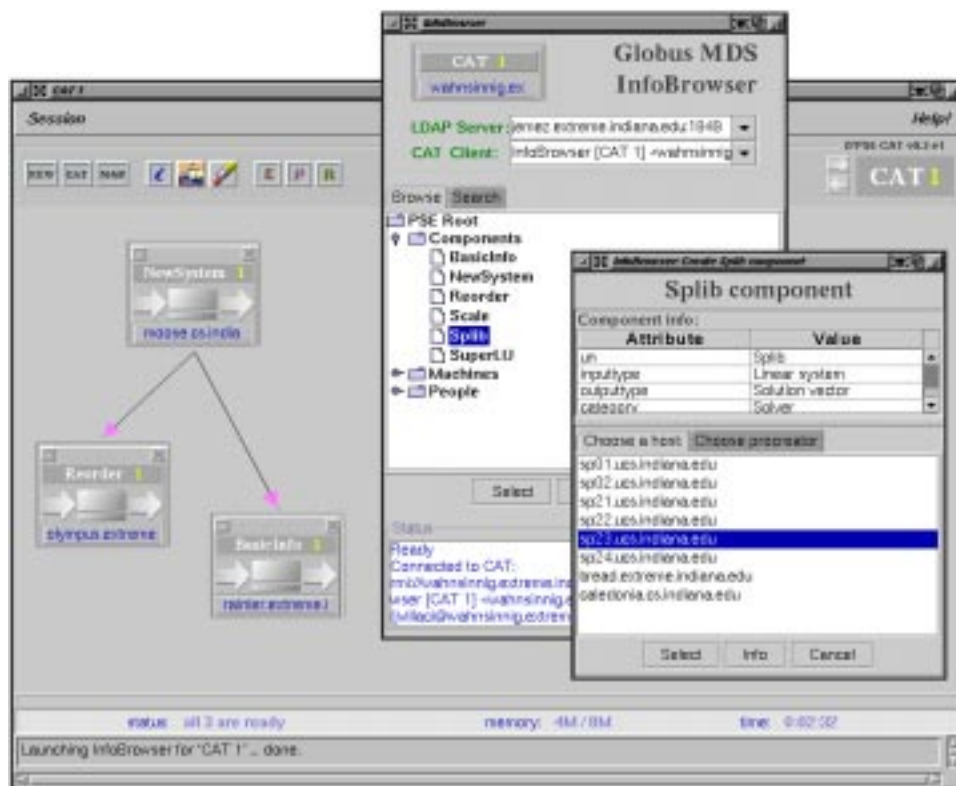


FIG. 3. *CAT workspace with InfoBrowser*

The **InfoBrowser** is used to locate and gather information about components prior to instantiating them, and the **workspace** is where components are visually manipulated via their graphical icons after they are instantiated. The events occurring within the workspace comprise a CAT session. An example CAT session is shown in Figure 3.

Next, we describe the steps involved in creating and manipulating components via the CAT.

### 3.2 Component Creation

In order to create a component, the user first clicks on the “*i*” button on the CAT toolbar. This launches an instance of the RIS InfoBrowser. The user can then browse the RIS as a directory tree (as shown in Figure 3) or search it via a text-based form. In either case, the user then obtains a listing of components that have been registered with the RIS. Double clicking on a selected component brings up a list of machines on which the component may run. The user then selects a machine and procreation mechanism (e.g., “RSH”, “GRAM”, etc.). Alternatively, the user may further explore the details about a component or machine before deciding whether or not to instantiate it. Once the user is satisfied with the selection, the component is instantiated on the desired machine using the specified creation mechanism, and an icon representing the component appears in the CAT workspace.

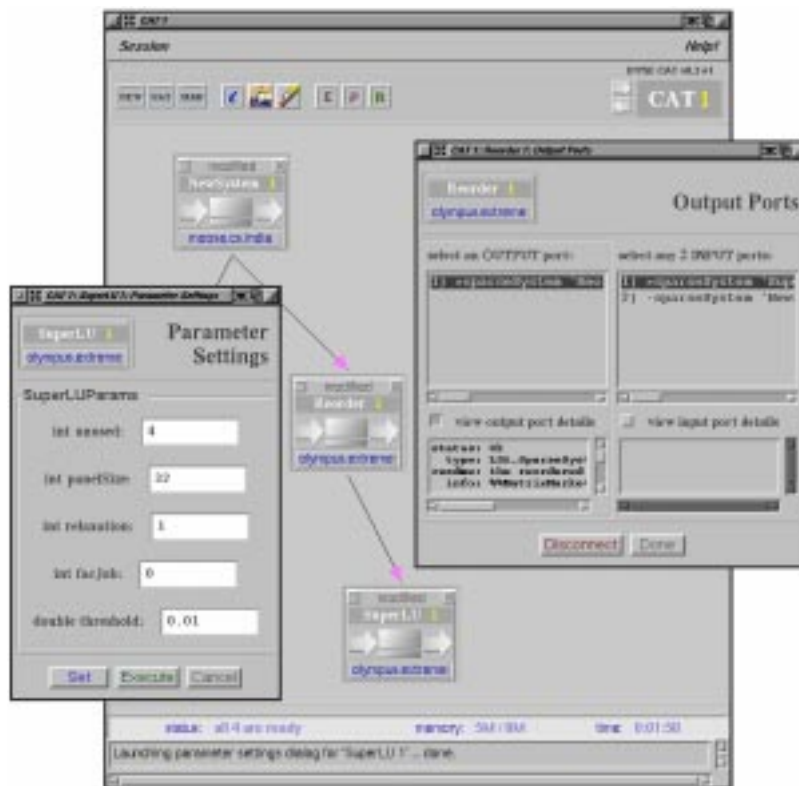


FIG. 4. Auto-generated parameter dialog and output port dialog

### 3.3 Component Interaction

After a component is instantiated, the user may interact with the component via its icon in the CAT workspace. Clicking on the “ $\Rightarrow$ ” areas of a component icon pops up a dialog box that prompts the user to connect/disconnect compatible input ports or output ports from other components running in the current workspace. Once a connection between components is established, an arrow indicating the direction of data flow (e.g., from an output port to one or more input ports) is drawn.

Clicking on the large center rectangle of a component’s icon brings up a dialog box which lets the user set internal parameters. Any input data to a component which is not from an input port is assumed to be an internal parameter. For these types of inputs, the CAT automatically generates a GUI with corresponding typed fields. At the bottom of a component’s parameter dialog box are three buttons to set the internal parameters, start the component running, or to cancel the operation.

Figure 4 shows an example of some component dialog boxes that can be launched via a component icon.

### 3.4 Other CAT workspace features

Several other features in the CAT workspace are worth mentioning since they aid in the useability of the CAT as *component management system*. Briefly, the CAT workspace provides additional facilities for:

- Capturing and saving an existing set of connected components as a *map*.
- Re-instantiating a saved map

- Saving the overall CAT configuration for later use or archiving.
- Attaching/detaching running components to/from different CATs via the registry
- Nesting CATs so that a group of components can be represented as a single component
- Rubberbanding components to form logical groupings or clusters of components
- Specifying data viewers for browsing output generated by a component

In sum, the CAT provides a framework and environment that allows users to easily create and compose components, and thereby rapidly form a running application, distributed across high performance computing resources.

### 3.5 Linear System Analyzer

The first application for the CAT system was to support the components from the Linear System Analyzer [3]. These components provide utilities for the analysis and solution to large, sparse linear systems of equations. The LSA components are grouped according to their primary functions:

- I/O components for reading systems from files, URLs, or running programs via sockets
- Filter components for manipulating the systems with re-orderings, scalings, or dropping of entries based on their relative sizes
- Solver components for actually solving the systems
- Information components for providing analysis of the linear systems.

Other than the input components, each has only one input port which takes a sparse linear system in compressed sparse row format. Other than the solver components (which produce a solution vector on their output ports), each sends a sparse linear system from its output port. The CAT allows multiplexing of output ports so the output of one component (such as a reordering) can be sent to several others (such as solvers), allowing a side-by-side comparison of solution strategies.

A typical LSA scenario is shown in Figure 5.

At the top, a NewSystem component has read in a linear system from a file. That system is fed to an iterative solver package (Splib 1), a matrix scaling, an informational component, a matrix reordering, and a sparse direct solver (SuperLU 1). The scaled system is sent to another instantiation of the iterative solver package (Splib 2), so that the results of solving the system with and without scaling can be compared by looking at the results of Splib 1 and Splib 2. Continuing this way, a user can dynamically explore and test various solution strategies for sparse systems of equations.

In this example, the sparse system is so large that only one will fit on any given machine. By distributing the components amongst machines (here, nodes of an IBM SP2) large numbers of strategies can be compared with the distributed solutions occurring in parallel, providing a latitudinal study between solution methods for a fixed linear system. If another linear system is read into the NewSystem component, it can be automatically propagated down through the tree of connected components, providing a longitudinal study between linear systems for a fixed component.

A key goal of any problem-solving environment is shielding the user from unnecessary details while providing any information needed by expert users - including performance data and solution quality as well as the solution itself. Clicking the smaller rectangle on a component's icon brings up a Web browser displaying an HTML page that has been automatically generated by the CAT. As shown in Figure 6, the web page contains links

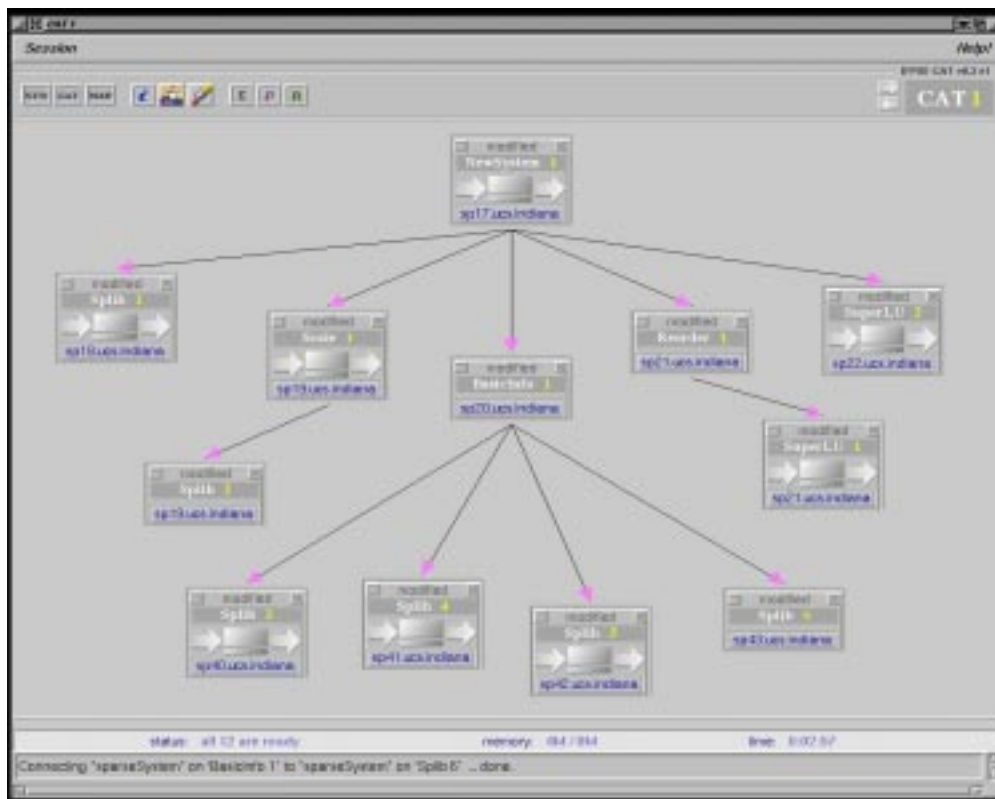


FIG. 5. CAT applied to a Linear System Analyzer (LSA) problem

to a dynamically generated file describing the machine (`sp19.ucs.indiana.edu` in this case), and for each linear system fed through the component, an “execution event” with summary data and links to any local files the component created. In this way the CAT allows numerical linear algebraists to quickly create and analyze solution strategies.

### 3.6 HPC Example

Another example of CAT applications was demonstrated at SC’98 [7]. In addition to the LSA components, this application had a component encapsulating a parallel industrial finite element code for mold-filling applications (CASTVIEW) [4], components encapsulating parallel linear solvers such as AZTEC [5], and a graphical component representing an immersive 3D visualization environment (VU) [6]. The CAT framework started up CASTVIEW running on SGI Origin 2000s at Indiana University, VU running on a workstation, an ImmersaDesk, and a CAVE at the conference in Orlando, and linear solver components running on several local multiprocessor SGIs at the conference. Using distributed Grid resources was necessary for this application. The finite element code simulated the mold-filling of a Canadian Pratt&Whitney engine part, with 2.55 million elements and 16 Gbytes of memory - the full capacity of the SGI Origin 2000s. Furthermore, the visualization had to be run where the viewers were located.

## 4 Conclusions

The CAT system provides a composition tool for the global Grid of HPC resources. It extends the Grid model to include distributed software components as well as hardware

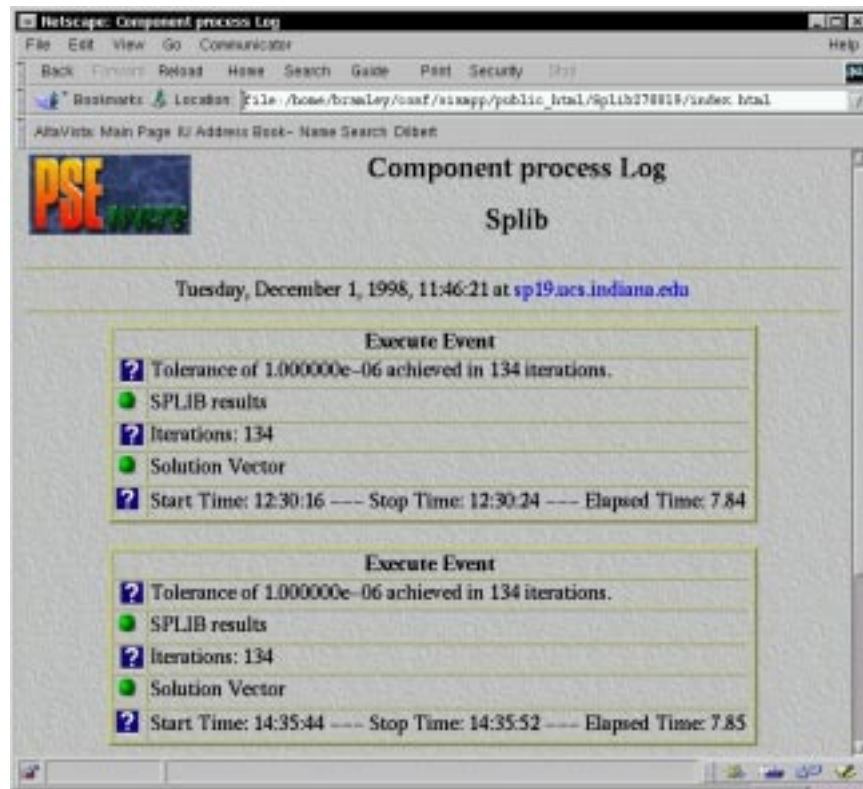


FIG. 6. Sample HTML page automatically generated for a CAT component

components, and shows this model of scientific computing is both conceptually and practically a powerful one. The CAT is based on a model of software components interacting as peers via ports dynamically linked with HPC networks. The CAT has succeeded in supporting an ambitious industrial application driving state of the art immersive 3D visualization. Such testing has proved invaluable in the refinement of the CAT design and has moved the CAT further towards its goal of providing an effective framework and toolset for Grid programming.

## References

- [1] F. BREG, S. DIWAN, J. VILLACIS, J. BALASUBRAMANIAN, E. AKMAN, AND D. GANNON, *Java RMI performance and object model interoperability: Experiments with Java/HPC++*, Concurrency and Experience, (1998). Presented at 1998 ACM Workshop on Java for High-Performance Network Computing.
- [2] I. FOSTER AND C. KESSELMAN, *Computational Grids*, Morgan-Kaufmann, 1998.
- [3] D. GANNON, R. BRAMLEY, T. STUCKEY, J. VILLACIS, J. BALASUBRAMANIAN, E. AKMAN, F. BREG, S. DIWAN, AND M. GOVINDARAJU. U, *Component architectures for distributed scientific problem solving*. Special Issue on Languages for Computational Science and Engineering.
- [4] F. ILLINCA, J.-F. HÉTU, AND R. BRAMLEY, *Simulation of 3-d mold-filling and solidification processes on distributed memory parallel architectures*. Proceedings of International Mechanical Engineering Congress & Exposition.
- [5] S. A. HUTCHINSON, J. N. SHADID, AND R. S. TUMINARO, *Aztec Users' Guide: Version 2.0*, Sandia National Laboratory, 1998. <http://www.cs.sandia.gov/CRF/aztec1.html>.

- [6] B. OZELL AND R. CAMARERO, *A configurable visualization environment*, In proceedings of 34th Aerospace Sciences Meeting and Exhibit. AIAA-96-0047. Reno, Nevada, January 1996. Also see <http://www.cerca.umontreal.ca/vu/welcome.html>.
- [7] CAT DEVELOPMENT TEAM, *SuperComputing'98 HPC Challenge*, 11 November 1998. <http://www.extreme.indiana.edu/sc98/sc98.html>
- [8] SUN MICROSYSTEMS, *Java Beans*, 1998. <http://java.sun.com/beans>
- [9] F. BREG, D. GANNON, *Compiler Support for an RMI implementation using Nexus-Java*, TR 500, Computer Science Dept., Indiana University, December 1997. Also see <http://www.extreme.indiana.edu/hpjava/NEXUSRMI/index.html>
- [10] I. FOSTER, C. KESSELMAN, ET. AL., *Globus Project*, 1998. <http://www.globus.org>
- [11] D. GANNON, P. BECKMAN, E. JOHNSON, AND T. GREEN, *Compilation Issues on Distributed Memory Systems*, chapter 3, In HPC++ and the HPC++Lib Toolkit, Springer-Verlag 1997. Also see <http://www.extreme.indiana.edu/hpc++/index.html>
- [12] PSEWARE RESEARCH GROUP, *Pseware home page*, 1 October 1997. <http://www.extreme.indiana.edu/pseware/>.