



a net-centric run-time support environment. Moreover, we present the realization of this framework for designing a prototype MPSE (GasTurbnLab) for supporting simulations needed for the design of efficient gas turbine engines.

•  
•  
•  
•  
•  
•  
•

## TABLE OF CONTENTS

<b>SECTION 1: INTRODUCTION .....</b>	<b>1</b>
<b>SECTION 2: MPSES - DEFINITIONS AND RESEARCH ISSUES.....</b>	<b>2</b>
SECTION 2.1: PSES AND MPSES.....	2
SECTION 2.2: THE RESEARCH ISSUES .....	3
<b>SECTION 3: THE GAS TURBINE ENGINE MULTIDISCIPLINARY APPLICATION .....</b>	<b>4</b>
<b>SECTION 4: GASTURBNLAB: A PROTOTYPE MPSE FRAMEWORK FOR GAS TURBINE ENGINE SIMULATIONS .....</b>	<b>5</b>
SECTION 4.1: FUNCTIONAL SPECIFICATIONS OF GASTURBNLAB .....	6
SECTION 4.2: ENABLING TECHNOLOGIES & SOFTWARE INFRASTRUCTURE .....	8
SECTION 4.3: GRAPHICAL USER INTERFACE .....	8
SECTION 4.4: MIDDLEWARE .....	9
<b>SECTION 5: GASTURBNLAB APPLICATION SOFTWARE .....</b>	<b>10</b>
<b>SECTION 6: ARCHITECTURAL OVERVIEW OF THE PROPOSED MPSE FRAMEWORK.....</b>	<b>12</b>
<b>SECTION 7: CONCLUSION.....</b>	<b>16</b>
<b>BIBLIOGRAPHY.....</b>	<b>17</b>
<b>APPENDIX A.....</b>	<b>22</b>
SOFTWARE AVAILABILITY .....	22
<b>APPENDIX B.....</b>	<b>23</b>
INTERFACE RELAXATION AND INTERPOLATION IMPLEMENTATION.....	23
<i>Functional description</i> .....	23
<i>interpolation Implementation Notes</i> .....	23
<i>Interface Relaxation Code</i> .....	24
<i>Interpolation Code</i> .....	25
<b>APPENDIX C.....</b>	<b>27</b>
IMPLEMENTATION OF A JAVA WRAPPER AROUND A LEGACY LINEAR SOLVER .....	27
<i>Java Driver Code</i> .....	27
<i>C Wrapper Code</i> .....	30
<b>APPENDIX D.....</b>	<b>32</b>
SAMPLE XML-FORMAT USED IN THE DATABASES .....	32
<b>APPENDIX E.....</b>	<b>33</b>
RESOURCE CHARACTERISTIC EVENTS.....	33
<i>Host Events Table</i> .....	33
<i>Network Performance Events Table</i> .....	34

---

## SECTION 1: INTRODUCTION

---

It is predicted that in the next century, the available computational power will enable any one with access to a computer to find an answer to any question that has a known or effectively computable answer. The recently proposed concept of problem solving environments (PSEs) [20][23] promises to contribute toward the realization of this prediction for physical modeling and to provide students, scientists, and engineers with environments that allow them to spend more time doing science and engineering rather than computing.

The predicted growth of computational power and network bandwidth suggests that computational modeling and experimentation will be one of the main tools in big and small science. In this scenario, computational modeling will shift from the current single physical component design to the design of a whole physical system with a large number of components that have different shapes, obey different physical laws and manufacturing constraints, and interact with each other through geometric and physical interfaces. For example, the analysis of an engine involves the domains of thermodynamics (gives the behavior of the gases in the piston-cylinder assemblies), mechanics (gives the kinematics and dynamic behaviors of pistons, links, cranks, etc.), structures (gives the stresses and strains on the parts) and geometry (gives the shape of the components and the structural constraints). The design of the engine requires that these different domain-specific analyses interact in order to find the final solution. The different domains share common parameters and interfaces but each has its own parameters and constraints. We refer to these multi-component based physical systems as multi-physics applications (MPAs). The realization of the above scenario, which is expected to have significant impact in industry, education, and training, will require the development of new algorithmic strategies and software for managing the complexity and harvesting the power of the expected HPCC resources; it will require PSE technology to support programming-in-the-large and reduce the overhead of HPCC computing. The main research thrust in this area should be to identify the framework for the numerical simulation of multi-physics applications and to develop the enabling theories and technologies needed to support and realize this framework in specific applications. The MPSE is the software implementation of this framework. It is assumed that its elements are discipline-specific problem solving environments. The MPSE design objective is to allow the natural specification of multi-physics applications and their simulation with interacting PSEs through mathematical and software interfaces across networks of computational resources. In this document, we describe a software architecture for MPSEs and its implementation for an MPA related to the simulation of gas turbine engines.

This document is organized as follows Section 2 defines the concepts of PSE and MPSE and reviews the associated research issues. Section 3 presents the gas turbine engine MPA. Section 4 discusses an MPSE, referred throughout as GasTurbnLab, for the simulation of gas turbine engines. In section 5, we describe the application software infrastructure in the GasTurbnLab prototype. In section 6, we describe the architectural components for a generic MPSE framework, along with issues pertaining to the GasTurbnLab instantiation of this MPSE framework. We conclude our discussion in Section 7, with an analysis of the overall MPSE framework architecture and the major challenges in validating this architecture and its principle objectives through the implementation of the GasTurbnLab prototype.

---

## SECTION 2: MPSEs - DEFINITIONS AND RESEARCH ISSUES

---

In the following we define the PSE and MPSE concepts, and review the associated research issues.

### SECTION 2.1: PSEs AND MPSEs

***Domain Specific PSEs:*** Even in the early 1960s, scientists had begun to envision problem-solving computing environments not only powerful enough to solve complex problems, but also able to interact with users on human terms. The rationale of our research is that the dream of the 1960s will be the reality of the 21st: High performance computers combined with better algorithms and better understanding of computational science have put PSEs well within our reach.

*What are PSEs?* A PSE is a computer system that provides all the computational facilities needed to solve a target class of problems. These facilities include advanced solution methods, automatic selection of appropriate methods, use of the application domain's language, use of powerful graphics, symbolic and geometry based code generation for parallel machines, and programming-in-the-large. The scope of a PSE is the extent of the problem set it addresses. This scope can be very narrow, making the PSE construction very simple. Nevertheless, even what appears to be a modest scope can be a serious scientific challenge. For example, we have created a PSE for bioseparation analysis [11][27]. This has a narrow scope, but is still a complex challenge as we incorporate both a computational model and an experimental process supported by physical laboratory instruments. We are also creating a PSE called PDELab for partial differential equations (PDEs) [60]. This is a far more difficult area than bioseparation and the resulting PSE will be less powerful (less able to solve all the problems posed to it), less reliable (less able to guarantee the correctness of results), but more generic (more able to parse the specifications of many PDE models). Nevertheless, PDELab will provide a quantum jump in the PDE solving power delivered into the hands of the working scientist and engineer.

*What are the PSE related research issues to be addressed?* A substantive research effort is needed to lay the foundations for building PSEs. This effort should be directed towards i) a PSE kernel for building scientific PSEs [62], ii) a knowledge based framework to address computational intelligence issues for PDE based PSEs [28][35], iii) infrastructure for solving PDEs [29][30][31][59][61], and iv) parallel PDE methodologies [13][38][40][63][64][65] and virtual computational environments [17][34][68].

***MPSEs for prototyping of physical systems:*** *If PSEs are so powerful, what then is an MPSE?* In simple terms, an MPSE is a framework and software kernel for combining PSEs for tailored, flexible multidisciplinary applications. A physical system in the real world normally consists of a large number of components that have different shapes, obey different physical laws and manufacturing/design constraints, and interact through geometric and physical interfaces. Mathematically, the physical behavior of each component is modeled by a PDE or ODE system with various formulations for the geometry, PDE, ODE, interface/boundary/linkage and constraint conditions in many different geometric regions. It is difficult to imagine creating a monolithic software system to accurately model such a real problem with complicated artifacts such as the turbo engine, which has literally hundreds of odd shaped parts and a dozen physical phenomena. Therefore, one needs an MPSE mathematical/software framework which, first, is applicable to a wide variety of practical problems, second, allows for software reuse in order to achieve lower costs and high quality, and, finally, is suitable for some reasonably fast numerical methods. Most physical systems and manufactured artifacts can be modeled as a mathematical network whose nodes represent the physical components in a system or artifact. Each node has a mathematical model of the physics of the component it

represents and a solver agent for its analysis. Individual components are chosen so that each node corresponds to a simple PDE or ODE problem defined on a regular geometry.

## SECTION 2.2: THE RESEARCH ISSUES

*What are the mathematical network methodologies required? What are the research issues?* There exist many standard, reliable PDE/ODE solvers that can be applied to these local node problems. In addition, there are nodes that correspond to interfaces (e.g. ODEs, objective functions, relations, common parameters and their constraints) that model the collaborating parts in the global model. Moreover, the analysis of an artifact changes through time, thus some of the interfaces appear and disappear during the analysis session. To solve the global problem, we let these local solvers collaborate with each other to relax (i.e., resolve) the interface conditions. An interface controller or mediator agent collects boundary values, dynamic/shape coordinates, and parameters/constraints from neighboring subdomains and adjusts boundary values and dynamic/shape coordinates to better satisfy the interface conditions. Therefore, the network abstraction of a physical system or artifact allows us to build a software system that is a network of collaborating well-defined numerical objects through a set of interfaces. Some of the theoretical issues of this methodology have been addressed in [44], [46] and [47] for the case of collaborating PDE models. The results obtained so far verify the feasibility and potential of network-based prototyping.

*What are the software methodologies for implementing the mathematical network? What are the research issues?* A successful architecture for PSEs requires heavy reuse of existing software within a modular, object oriented framework consisting of layers of objects. The kernel layer integrates those components common to most PSEs or MPSEs for physical systems. We observe that this architecture can be combined with an agent-oriented paradigm and collaborating solvers [16] to create MPSE as a powerful prototyping tool. MPSEs must exploit and build on the new technologies of computing. By the time MPSEs are operational, the advances in computing power and the communication infrastructure will allow ubiquitous high performance computing, i.e., every where by every one. The designs for MPSE must be application and user driven. An MPSE must simultaneously minimize the effort and maximize the solution power delivered to researchers, engineers and scientists, students, and trainees. We should not restrict our design just to use the current technology of high performance computers, powerful graphics, modular software engineering, and advanced algorithms. We see MPSE as delivering problem solving services over the Net. This viewpoint leads naturally to collaborating agent-based methodologies. This, in turn, leads to very substantial advantages in both software development and quality of service as follows. We envision that a user of MPSE will receive at his location only the user interface. Thus, the MPSE server will export to the user's machine an agent that provides an interactive user interface built on top of the standard services of the Net. The bulk of the software and computing is done at the server's site using software tailored to a known and controlled environment. The server site can, in turn, request services from specialized resources it knows, e.g., a commercial PDE solver, a proprietary optimization package, a 1000 node supercomputer, an ad hoc collection of 122 workstations, a database of physical properties of materials. Each of these resources is contacted by an agent from the MPSE with a specific request for problem solving or information service. Again, all this collaboration is built on standard network services. All of this can be managed without involving the user (if s/he so desires), without moving software to arbitrary platforms, and without revealing source codes.

*What are the design objectives of an MPSE for physical system design? What are the research issues?* These mathematical networks can be very big for major applications. For a realistic turbine simulation, there are perhaps 100 million variables and many different time scales. This problem has very complex geometry and is very non-homogeneous. The answer (a data set that allows one to display an

accurate approximate solution at any point) is 20 gigabytes in size and requires about 10 teraflops to compute. This data set is much smaller than the computed numerical solution. The network of PDE solvers might have 10,000 subdomains and 35,000 interfaces. A software network of this type is a natural mapping of a physical system and simulates how the real world evolves. This allows the use of the software parts technology (object-oriented programming) that is the natural evolution of the software library idea. It allows software reuse for easy software update and evolution, things that are extremely important in practice. The real world is so complicated and diverse that we believe it is impractical to build monolithic, universal solvers for such problems. Without software reuse, it is impractical for anyone to create on his own a large software system for a reasonably complicated application. Each new automobile normally results in a new software system. Recreating such a system could easily take several months or years. In contrast, the execution time to perform the required computation might only be a few days. Notice that such a physical change usually corresponds to replacing, adding, or deleting a few nodes in the network with a corresponding change in interface conditions. These are simple manipulations on a network, which do not affect the rest of the system and can thus be easily done. In this application, each physical component can be viewed both as a physical object and as a software object. In addition, this mathematical network approach is naturally suitable for parallel computing as it exploits the parallelism in physical systems. One can handle issues like data partition, assignment, and load balancing on the physics level using the structure of a given physical system. Synchronization and communication are controlled by the mathematical network specifications and are restricted to interfaces of subdomains, which results in a coarse-grained computational problem. This is especially suitable for today's most advanced parallel supercomputer architectures. The network approach also allows high scalability. Realizing this MPSE technology requires research advances both in the general structure and implementation area and in more specific areas from the target applications. For example, we must design and create the tools that allow the MPSE agents to collaborate over the Net. We must create a flexible and general methodology for interfacing large and heterogeneous software systems. Following we propose a software framework for MPSEs supporting PDE based applications and realize it for a multi-physics application related to the simulation of gas turbine engines.

---

### SECTION 3: THE GAS TURBINE ENGINE MULTIDISCIPLINARY APPLICATION

---

The gas turbine engine is an engineering triumph. It has more than 1,300 parts with rotational speeds to 16,000 rpm for axial and 50,000 rpm for radial flow components. For aircraft applications, it operates with maneuver loads of up to 10g, with flow path pressures and temperatures to 40 atmospheres and 1400 F. The extreme complexity and high-performance requirements of aircraft gas turbines are illustrated in Figure 1. The important physical phenomena take place on scales from 10-1000 microns to meters. A complete and accurate simulation of an entire engine is enormously demanding; it is unlikely that the required computing power, simulation technology or software systems will be available in the next decade. The primary goal of the GasTurbnLab research project is to advance the state-of-the-art in very complex scientific simulations and their validation. Specifically, we consider simulating the compressor-combustor-turbine coupling in a gas turbine engine [21]. For this we plan to design and implement a MPSE, referred as GasTurbnLab, to study complex physical phenomena such as stall, surge and turbine blade fatigue. Figure 2 presents an abstraction of a MPA and the corresponding software infrastructure required. The hardware infrastructure assumed for these simulations and the implementation of MPSE consists of a computational grid involving a SP-2, 32 PC cluster running Solaris, and SGI Origin 2000 with 32 CPUs. In this study we will utilize the agent system *Grasshopper* that is MASIF (Mobile Agent System Interoperability Facilities Specification) standard compliant and runs on the top of CORBA [69]. Details of this implementation follow.

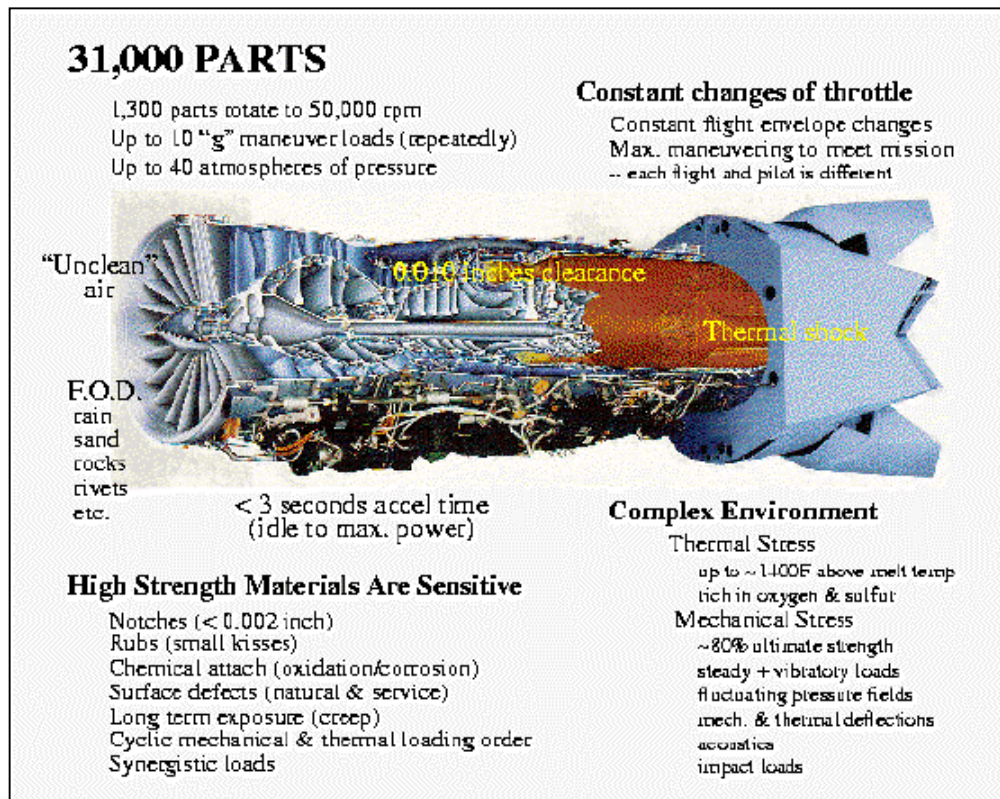


Figure 1: View of a gas turbine showing some of its detail, some of its operational characteristics, and the engineering methodologies involved in its design, simulation and construction

---

#### SECTION 4: *GASTURBNLAB*: A PROTOTYPE MPSE FRAMEWORK FOR GAS TURBINE ENGINE SIMULATIONS

---

In this section we describe the design of a MPSE framework that can be used to simulate complex multi-physics phenomena governed by PDE network models in general and the requirements of the GasTurbnLab MPSE in particular [20]. A network of distributed machines is assumed as the hardware infrastructure. The PDE simulations are often defined on geometric domains. Thus, the natural geometric boundaries or artificial geometric boundaries can be used to split the problem and the underlying simulation into many smaller sub-problems. Each sub-problem would then be solved independently, with mediator interactions along the boundaries for interface relaxation. Thus, the MPSE framework for PDE simulations must support domain decomposition with geometric objects, usage of a network of PDE solver agents, and interface relaxation. Our design goal in GasTurbnLab MPSE is to identify existing software solvers that can support this paradigm assuming that the application computational resources consist primarily of "legacy" code.

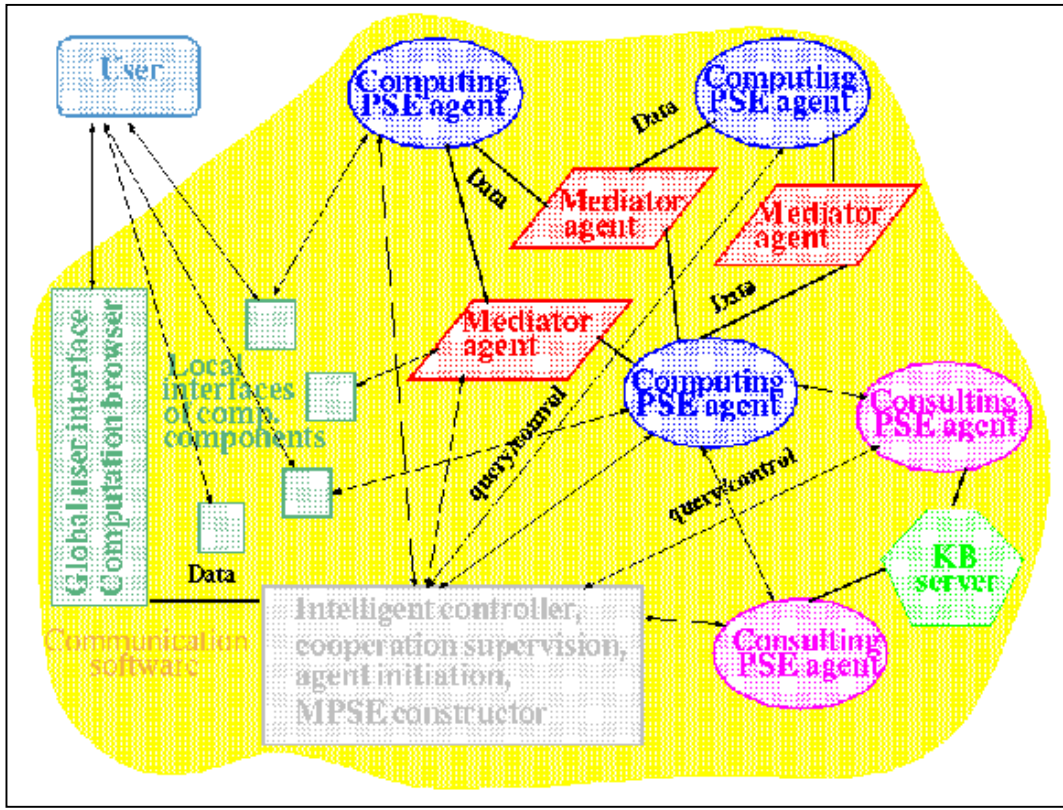


Figure 2: Functional view of a multidisciplinary PSE. The computations (and the major data exchange) are concentrated in the network of solver (PSE) and mediator agents. The solver agents communicate with the recommender ones through queries to obtain “advice” on computational parameters. The user interacts with the system through the global and local user interface, which send queries and receive replies from the various agents.

#### SECTION 4.1: FUNCTIONAL SPECIFICATIONS OF GASTURBNLAB

In the case of PDE simulations, the MPSE framework user interface is driven by the underlying geometric modularity of the problem. The geometry is assumed to have a root node for the target object and the user is allowed to subdivide it in multiple ways, resulting in a hierarchy of geometrical objects. The interface would allow user-access to relevant data associated with the geometric objects at every level.

This geometric domain decomposition of the target simulation object defines a network of PDE problems. On each subdomain, a PDE problem models the physics on that geometric object (domain). Each subdomain has some neighbors and possibly, some fixed boundaries. If each neighborhood connection is represented by an arrow, we get an abstraction of a network of PDE problems. Since the PDEs on each domain are usually not the same, these represent a composite PDE problem. The MPSE framework maps the network of PDE problems resulting from a user-specified partitioning, onto a set of computational agents on a pre-specified collection of machines. This resource allocation will be done in an optimal manner to minimize the communication overhead between computational agents of neighboring subdomains. However, the MPSE framework interface would allow the user to manipulate this mapping to achieve a custom resource allocation.

Under the assumption that any single PDE problem of the composite problem can be solved exactly, the interface relaxation mathematical technique will be used to solve the composite PDE problem. The interface relaxation methodology is based on the iteration shown in Figure 3. An implementation of this methodology is listed in

Appendix B for the gas turbine engine simulation MPSE.

In the GasTurbnLab MPSE, the initial target object would be the entire gas turbine engine. Thus, a simulation in the GasTurbnLab MPSE consists of a user-specified set of geometrical objects that partition the engine and a corresponding network of PDE solver agents that collaborate to find a solution for the composite PDE problem. The geometrical objects that partition the engine may be hierarchical, resulting in a corresponding set of hierarchical computations in an asynchronous simulation process.

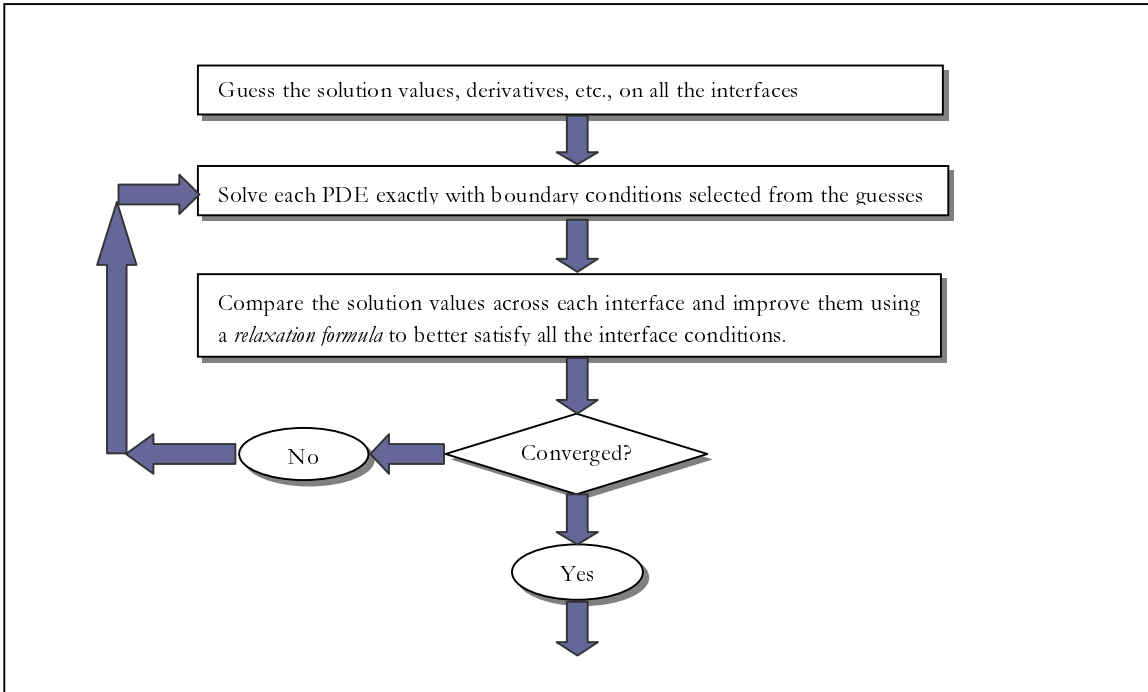


Figure 3: Interface relaxation iteration.

#### SECTION 4.2: ENABLING TECHNOLOGIES & SOFTWARE INFRASTRUCTURE

Utilizing existing technology and legacy software is an important goal in the design of this MPSE framework and its prototype implementation, GasTurbnLab. The MPSE framework is built across three main architectural components - *the user interface layer*, *the middleware*, and *the computational software infrastructure* layer. The IRIS Explorer application builder and visualization system [70] will be used for the MPSE framework user interface and the middleware component will be based on the Grasshopper mobile agent platform [69]. The computational infrastructure is dependent upon the MPSE's target class of simulation problems. This computational application software infrastructure is discussed in section 5. Figure 4 depicts these architectural layers and their major constituents for the GasTurbnLab PSE.

#### SECTION 4.3: GRAPHICAL USER INTERFACE

The IRIS Explorer system is a toolkit for data visualization and uses a dataflow paradigm. An interface is built by creating requisite modules and wiring them together via Explorer's map editor. The resulting "map" may be saved for reuse or edited on the fly using modules listed in the Module Librarian. The Module Librarian contains modules and maps organized into categories.

The connections in an Explorer map depict the flow of data between modules and act as module triggers. Modules have input ports and output ports, interactively controllable parameters and the ability to execute on different machines on a network. A module is activated when all its input ports are triggered. IRIS Explorer allows modules written in C/C++ to issue scripting commands through a scripting API. The SKM language with Lisp-like syntax is used to create these scripts for the Explorer command interface.

Molding the proposed MPSE framework design into IRIS Explorer's dataflow steering paradigm along with its visual programming interface will be a significant implementation challenge. Self-contained, compact modules with the requisite, well-defined data flow interfaces will have to be implemented to achieve a seamless PSE interface.

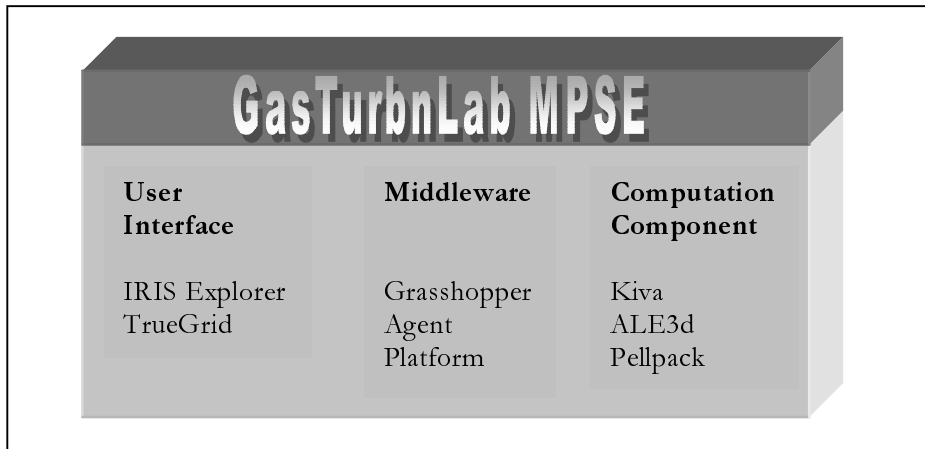


Figure 4: Major components of the GasTurbnLab MPSE.

The IRIS explorer interface will provide access to all the MPSE framework components that are potentially user-steerable, including problem specification and solution visualization. The GasTurbnLab prototype implementation will utilize the TrueGrid software tool [71] for problem specification and IRIS Explorer's data visualization system for post-simulation solution analysis. The TrueGrid domain specification tool will be incorporated into the framework as a self-contained Explorer module. The IRIS Explorer user interface will launch the computational module, which interacts with the underlying Grasshopper agent platform. Grasshopper's graphical monitoring tool will be used within the computational module to view and monitor the underlying agent interactions in the simulation process for possible computational steering. Once the computational module completes its simulation task, control will be returned to the Explorer interface for the solution visualization and analysis phases. The IRIS Explorer based user interface for the PSE framework is described in detail in Section 6.

#### SECTION 4.4: MIDDLEWARE

The MPSE framework will use the Grasshopper Distributed Agent Environment (DAE) as middleware to facilitate the agent-based computational simulation paradigm. The Grasshopper mobile agent platform is MASIF compliant (the first mobile agent standard of OMG), and is built on top of a distributed processing environment. It is implemented in Java to achieve platform interoperability and offers a range of communication protocols for remote interaction (IIOP, RMI or

plain socket connections). The DAE is composed of *regions*, *places*, *agencies* and different types of *agents* that may be either stationary or mobile. Agencies are the actual runtime environments for the agents and hence at least one agency should be running on each host machine. A place provides a functional grouping within an agency. Regions facilitate the management of the distributed components with a region registry used to maintain information about all components in a specific region.

During their lifecycle, Grasshopper agents may be in one of the following states: *active*, *suspended* or *deactivated*. Grasshopper agents may be either mobile or stationary. Unlike traditional mobile code that usually features remote execution (where the program is sent *before* execution), mobile agents can migrate *during* execution. Integrating mobile agent technology and client/server or peer-to-peer communication technology yields many possible agent interaction scenarios:

- Remote communication
- Client agent migration to a traditional server
- Server agent migration to a traditional client
- Dual peer agent migration to an intermediate location plus local communication
- Single peer agent migration to a convenient intermediate location plus remote communication

Due to the importance of legacy code usage and the problems inherent to legacy code migration, the MPSE framework will utilize a combination of these interactions.

The Grasshopper communication service provides the *means* for location transparent, inter-agent communication with multi-protocol facilities such as IIOP, RMI and TCP/IP sockets. However, it does not specify the *ways* of communication with a specific agent language. RMI and socket connections can be made secure with SSL (Secure Socket Layer) protection. Additionally, Grasshopper makes use of X.509 certificates to ensure confidentiality, integrity and proper authentication. For access control, Grasshopper uses the JDK 1.2 security mechanisms. Grasshopper provides a persistence mechanism for agents and offers a standard array of communication modes – synchronous, asynchronous, dynamic and multicast.

The MPSE framework will initially use RMI and plain sockets for its agent interactions. A proprietary language based on either XML or an existing agent communication language will be used for agent communication. Security issues in the MPSE framework will be addressed at all levels and the realization of the MPSE security framework will include the mechanisms available in Grasshopper.

---

## SECTION 5: GASTURBNLAB APPLICATION SOFTWARE

---

The computational infrastructure in the MPSE determines its target class of problems. The proposed MPSE framework provides the architecture and model infrastructure for an agent-based simulation MPSE and facilitates a straightforward incorporation of computational code to GasTurbnLab. The MPSE framework design takes into consideration the possibility of legacy code in the computational component, as in the case of GasTurbnLab. The introduction of legacy code infuses a certain level of intractability into the computational agent design since we cannot assume that legacy software can be inserted within a *mobile* agent.

The computational software infrastructure in GasTurbnLab will consist of Ale3D, Kiva-3V, and PELLPACK code modules and interface relaxation code implemented in either C/C++ or Java. ALE3D is an advanced CFD software module targeted to gas turbine simulation. It is large, with

about 200,000 lines of code. KIVA-3V is an advanced combustion-simulation package with about 500,000 lines of code. PELLPACK is a versatile PSE for PDE problems, encapsulating many PDE solvers and graphical support tools. It has more than a million lines of code.

ALE3D, KIVA-3V and most of PELLPACK's PDE solvers are implemented in Fortran. There are two approaches to incorporate this legacy Fortran code into the PSE framework's Java-based agent structure.

1. Inserting the Fortran-based code within Java wrappers as stationary agents. This can be achieved with JNI (Java Native Interface). Figure 5 illustrates the encapsulation technique within a stationary agent.
2. Inserting the Fortran-based code within C/C++ wrappers as servers. They can then be accessed as local servers by client agents. Figure 6 illustrates the legacy code embedded server and the client agent interaction.

An implementation of a Java wrapper around a legacy linear solver is listed in Appendix C.

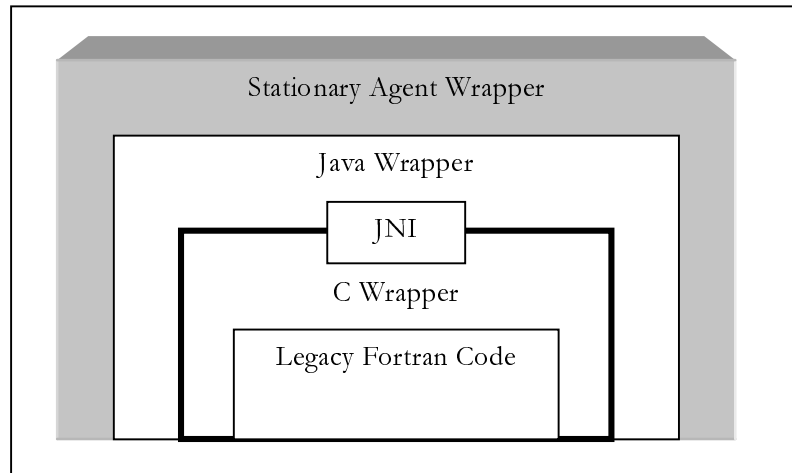


Figure 5: Anatomy of a legacy code embedded stationary agent.

The advantage of the first approach is that it fits elegantly into the proposed computational scenario. However, the legacy code's inherent interface requirements may complicate the use of JNI and result in a very restricted wrapper. Furthermore, if the wrapper becomes very large and involves complicated programming with many Fortran, C and Java code interactions, this would not be in the best the best approach. The second option would then be easier to implement, albeit introducing additional necessities such as a communication protocol between the legacy-code-wrapped servers and client agents. Hence, choosing a specific approach should be done on a case-by-case basis, depending on the legacy software. Both approaches should optimize memory and bandwidth usage with attention to performance and robustness. The MPSE framework design to allow legacy code incorporation based on either of these two approaches.

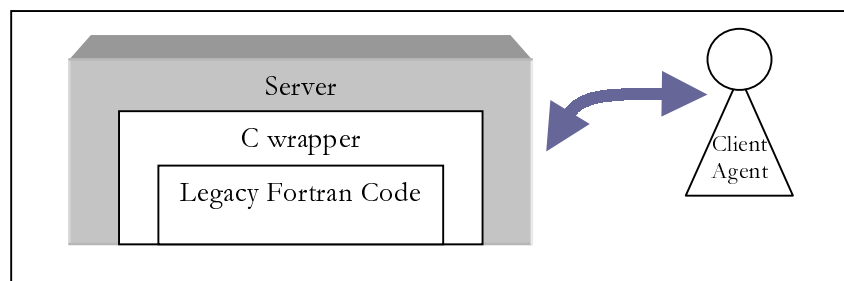


Figure 6: The Client/Server approach for legacy code encapsulation.

---

**SECTION 6: ARCHITECTURAL OVERVIEW OF THE PROPOSED MPSE FRAMEWORK**

---

In this section, we present an overview of the agents and other components contained in the MPSE framework. We discuss the overall generic architecture (Figure 7) and include details in the case of a specific MPSE (GasTurbnLab) implemented using this framework.

The graphical user interface of the PSE framework mainly comprises the problem specification, dispatcher and compute modules. These will be implemented as IRIS Explorer modules. The dispatcher and compute modules interact with specific agents in the underlying Grasshopper platform. These agents enable the actual simulation computations in the PSE.

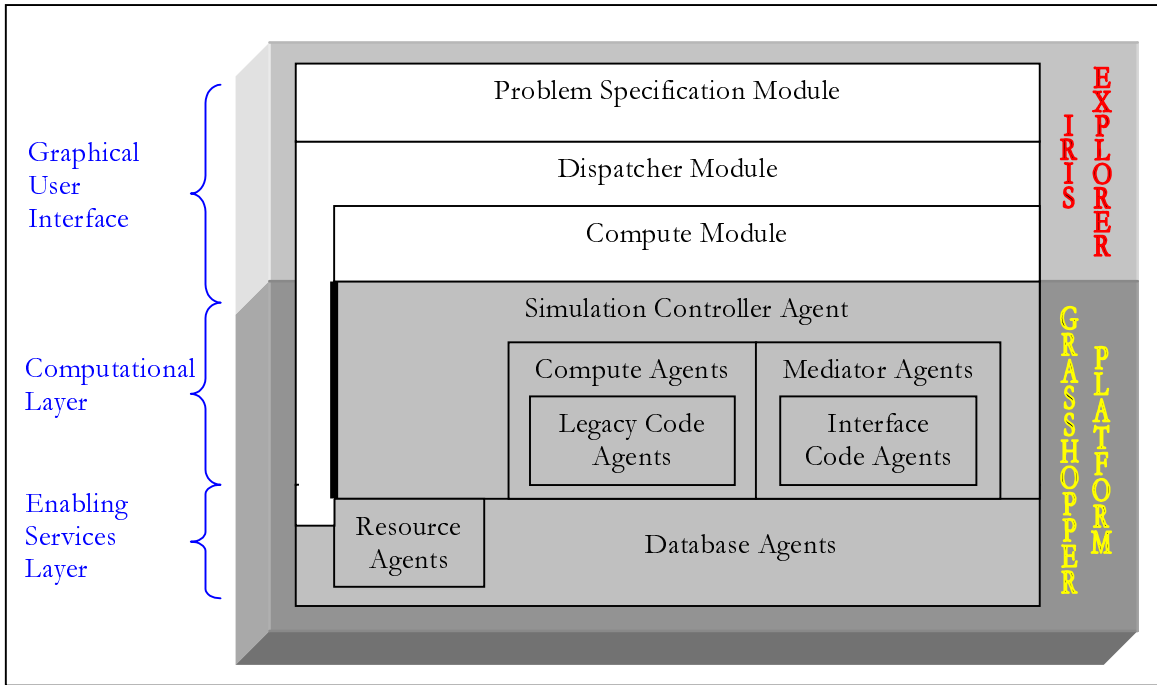


Figure 7: The MPSE framework architecture.

**Enabling Services Layer:** The Grasshopper distributed agent environment runs on all the hosts of the networked computational grid. Each host agency has an active *DataBase Agent* (DBA) and an

active *Resource Agent* (RA). They are implemented as stationary Grasshopper agents. The DBA agent will control the local database on the host. It will have sole responsibility of authenticating data entry, update and retrieval requests. In addition, this agent may have the capability to respond to properly authenticated HTTP protocol requests, enabling Web-based data retrieval and visualization. The data in these databases will be stored in an XML format based on a proprietary DTD (Document Type Definition). Such a specification will only be applicable to meta-data. For instance, the linear system elements would not be stored in XML-format. Instead, a pointer (URI – Universal Resource Identifier) to the linear system data will be specified in XML-format. Thus, the PSE framework does not impose any requirements on the linear system data itself, and it may be stored in any format determined by the underlying legacy computational software. A linear system stored in this format is listed in Appendix D. The RA agent will monitor execution performance and gather local machine load and network congestion information. It will maintain a local resource database along with other requisite logs. The local RA will synchronize its resource information with resource agents on other hosts. Thus, each RA will have access to dynamic network information such as load, congestion and machine reachability. The local RA may be queried for the latest resource data or it may be instructed to provide updates to specific remote agents. The update frequency can be periodic or triggered by the occurrence of certain *Resource Characteristic events* (RC events), such as the local host processor load reaching a particular level. The RA may maintain the resource database as part of the local database in conjunction with the DBA agent. The relevant RC events are listed in Appendix E. This event model for resource monitoring will facilitate the incorporation of various resource management tools and techniques in the upper layers of the architecture. For instance, the compute layer may use the resource characteristic events to implement a range of load balancing models.

**User Interface Layer:** As described in section 4, the *Problem Specification (PS) module* with the embedded TrueGrid tool will be used to specify the root domain and its decomposition. The formatted output from this module will be directed to the *dispatcher module*. The dispatcher will distribute the partitioned data to the local databases of selected hosts on the available computational grid. It will have the capability (i.e., a set of allocation algorithms) to select the physical host locations for each subdomain computational agent and the mediators based on information provided by the local resource agent. This allocation will attempt to optimize network connectivity and host load. The dispatcher will have a graphical interface to display its actions, allowing the user to override its decisions or modify the allocation algorithm parameters. Upon successful completion of the data distribution, the dispatcher module will generate a *host allocation table* as its output. The dispatcher module may also be wired in an Explorer map for other data distribution tasks. For instance, it could be utilized for a distributed, collaborative solution analysis session. The output from the dispatcher module will usually be directed to the *compute module (CM)*. The CM will control the launch and execution of the computational agents. It will monitor the simulation process until the user-specified stopping condition is reached. The output of the CM would be the simulation problem solution. Figure 8 illustrates this module and the other main Iris Explorer modules of the PSE framework in a wired map.

**Computational Layer:** The primary “workers” within the CM are the *Compute Agents (CA)* and *Mediator Agents (MA)*. The CA, when activated, reside on each target host with a single agent per domain partition. It is feasible, although not desirable, for a host agency to have more than one active compute agent during a simulation process (implying more than one domain partition having been assigned to the host). The compute agents are implemented as mobile Grasshopper agents. The

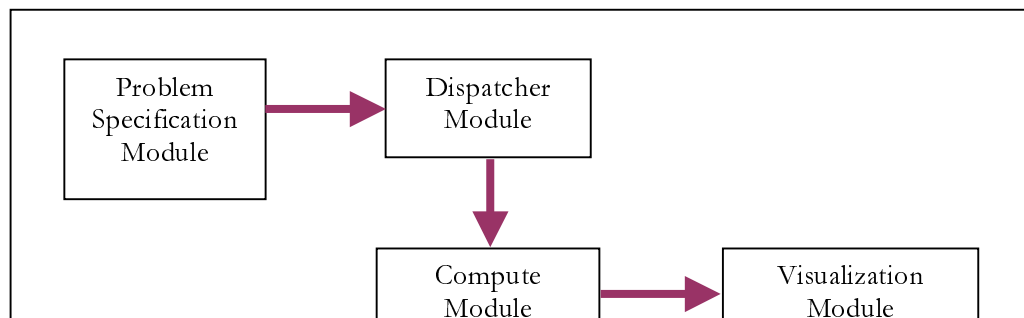


Figure 8: Iris Explorer module map for the PSE framework interfaces

mediator agents, when activated, may reside on a target host with a domain partition or on an intermediate host in close proximity to two target hosts with neighboring domain partitions. The mediator agents are also implemented as mobile Grasshopper agents. After describing the CM in detail, we will discuss the architectural technique that makes the compute and mediator agent mobility possible, even when the simulation computation has to be performed by legacy code.

The compute module realizes its task by launching a *Simulation Controller Agent (SCA)*. This agent controls the entire computational simulation process by monitoring the distributed compute agents and mediator agents on each host. The simulation controller interacts closely with the resource agents on the target hosts to ensure the dynamic integrity of the selected computational grid. This interaction may be either via periodic updates or via RC event occurrences. For instance, if a particular host connection deteriorates, the simulation controller agent may instruct the corresponding compute agent to continue its computation after migrating to another host. If necessary, the simulation controller will inform the other relevant compute agents and mediator agents of the migration. However, since the Grasshopper environment supports location transparent communication for its mobile agents, depending on the agent communication implementation, such notification may not be required. Furthermore, for highly compute intensive simulations, the simulation controller may employ load-balancing techniques to redistribute the ongoing computations amongst the processors on the computational grid. The compute and mediator agent mobility makes this operation possible without disrupting or restarting the simulation computations. We propose a two-tiered agent/wrapper architecture to facilitate compute and mediator agent mobility within the PSE framework. The actual legacy codes (if any) for the compute agent will be encapsulated within a *Legacy Code Agent (LCA)*. The actual legacy codes (if any) for the mediator agent will be encapsulated within an *Interface Code Agent (ICA)*. This second tier of wrappers will exist transparently within the PSE framework. Thus, all other agents in the framework will interact solely with the compute and mediator agents and not the LC and IC agents. The LC and IC agents will communicate only with their corresponding compute and mediator agents. The possible agent interactions within the PSE framework are schematically depicted in Figure 9. Although we refer to these second tier components as agents, their actual implementation may be in the form of legacy code embedded servers (as described in above). For clarity, we will continue to refer to them as agents, irrespective of their possible implementation technique.

A compute agent may be required to migrate to another host for load balancing purposes. In this event, the simulation controller directs it to use a different LC agent. Since only the compute agent can physically migrate, it requests the LC agent to stop computation of the current iteration. It then migrates with the *last completed iteration* data to its next location. The compute agent then starts the next iteration computation with the new LC agent with its saved “last completed iteration” data. To make such mobility possible, the compute agent is required to always save the last completed iteration data. The mediator agent migration is also achieved in a similar manner.

A compute agent may be required to migrate to another host for load balancing purposes. In this event, the simulation controller directs it to use a different LC agent. Since only the compute agent can physically migrate, it requests the LC agent to stop computation of the current iteration. It then migrates with the *last completed iteration* data to its next location. The compute agent then starts the next iteration computation with the new LC agent with its saved “last completed iteration” data. To make such mobility possible, the compute agent is required to always save the last completed iteration data. The mediator agent migration is also achieved in a similar manner.

The LC and IC agent availability on the computational grid hosts will be recorded as part of the resource information in the PSE framework. Thus, the LC and IC agent locations will be considered by the allocation algorithms of the dispatcher module when assigning the partitioned domains to the computational grid hosts. This information will also be available to the load balancing algorithms in the simulation controller agent. The LC and IC agents may not be available on all the hosts of the computational grid. In such a situation, if a compute agent migration were triggered by load balancing requirements or network congestion, the agent would be moved to a location with an available LC agent in close proximity.

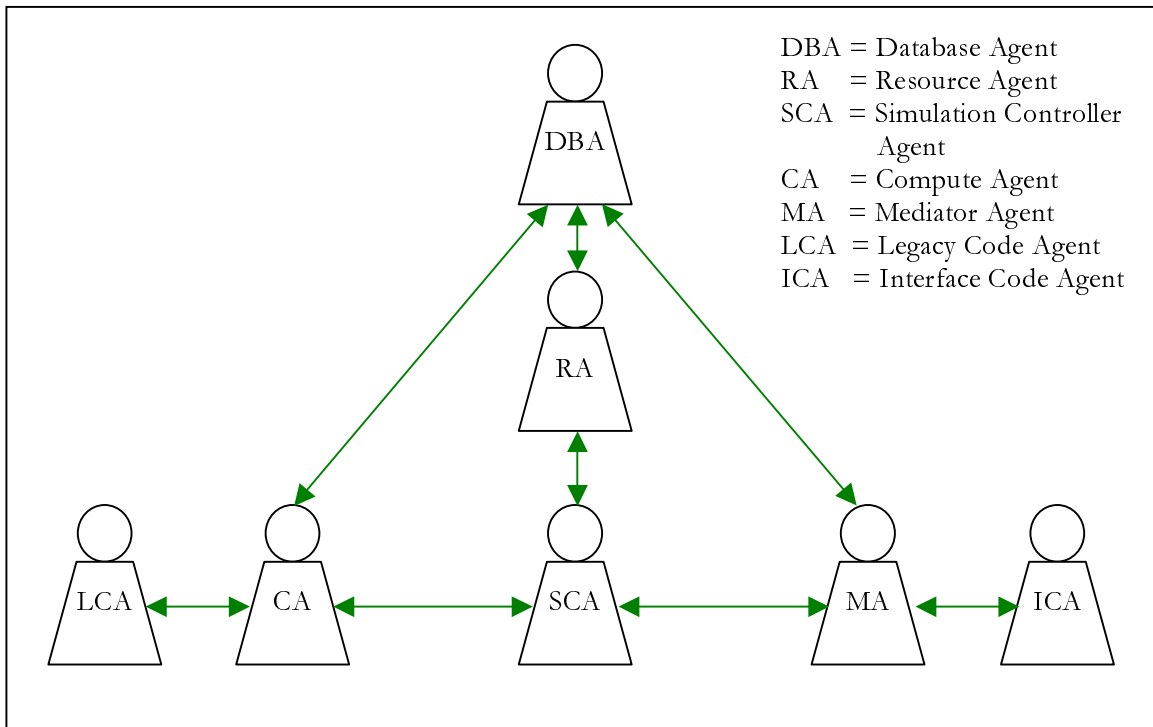


Figure 9: PSE framework agent interactions.

---

## SECTION 7: CONCLUSION

---

In summary, we describe an agent-based framework to build problem solving environments for large scale simulation tasks. This framework design is based on the geometric modularity approach for the simulation computations.

The MPSE framework uses the IRIS Explorer system as its front-end and the Grasshopper Agent Platform as its middleware infrastructure. We have proposed a layered architecture for the framework. This design incorporates the extensibility features of the Explorer system and the mobile agent features of the Grasshopper platform. The MPSE framework may be extended at the user interface level by wiring additional modules based on the Explorer model. Furthermore, the framework may be extended at the enabling services and computational levels by inserting new mobile or stationary agents to perform additional services or computations.

To facilitate legacy code incorporation, we propose a two-tiered agent/wrapper architecture for the computational agents in the PSE framework. This design allows the use of mobile agents with legacy computational code, promoting robustness and better performance for this class of simulation problems.

Optimum resource usage and management is an important goal for a distributed PSE. We facilitate these tasks with the Resource Characteristic event model in the enabling services layer. This design feature enables the implementation of load balancing techniques and optimization algorithms for memory and bandwidth usage.

The MPSE framework design does not specify the underlying database technology. Thus, the implementation may include an off-the-shelf database system or a custom-designed database. In either case, the database system needs to have an API that allows interaction with the MPSE framework's database agents.

The GasTurbnLab MPSE is a realization of the agent based MPSE framework for the simulation of gas turbines. The large body of legacy code needed for this simulation can be easily incorporated within the MPSE framework using the two techniques outlined in Section 5. A suitable load balancing algorithm can be implemented within the simulation controller agent for better distributed performance of the highly compute intensive simulations. The graphical user interface can be tailored appropriately with suitable problem specification modules that include tools such as TrueGrid and MeshTV. The GasTurbnLab MPSE implementation may contain a library of Explorer modules for such problem specification tools, including additional modules for different solution visualization tools. This would enable the scientist to customize the GasTurbnLab user interface with suitable pre- and post-processing modules for each target gas turbine simulation problem.

The proposed MPSE framework architecture is scalable, enabling it to be used to build very large scale, distributed problem solving environments for scientific simulations. It is also versatile and simple enough to be used to rapidly build prototype problem solving environments to analyze and validate mathematical techniques for interface relaxation. Thus, it would be a useful environment towards advancing the state-of-the-art in simulating complex physical phenomena.

---

## BIBLIOGRAPHY

---

1. Richard M. Adler, Emerging standards for component software, *IEEE Computer*, March (1995), 68-77.
2. R. Balling, J. Sobieszczanski-Sobieski, Optimization of coupled systems: A critical review of approaches, ICASE Report No. 94-100, December 1994, 30 pages.
3. K.C. Bernard, Ordering chaos: Supercomputing at the edge, in *Technology 2001: The future of Computing and Communications*, D. Leebaert, editor, MIT Press, Cambridge, MA, (1992).
4. J. A. Berninger, R. D. Whitley, X. Zhang and N.-H. L. Wang, A versatile model for simulation of reaction and nonequilibrium dynamics in multicomponent fixed-bed adsorption processes, *Computers in Chemical Engineering*, **15** (1991), 749-768.
5. R. Boisvert, The guide to available mathematical software advisory system, in *Intelligent Mathematical Software Systems* (E.N. Houstis, J.R. Rice, R. Vichnevetsky, eds.), North-Holland, 1990, 167-179.
6. L. Boloni and D.C. Marinescu, An Object-Oriented Framework for Building Collaborative Network Agents. Kluwer Publishers, 1999 (to appear).
7. J. M. Bradshaw, An introduction to software agents, in J. M. Bradshaw
8. Ed. Software Agents, MIT Press, 1997, pp. 3-46.
9. K. Brockschmidt, *Inside OLE 2*, Microsoft Press, Redmond, Washington (1994).
10. R.E. Burkart, Reducing the R&D cycle time, *Research Tech. Mgmt.*, **37** (1994), 27-32.
11. A.C. Catlin, M.G. Gaitatzes, E.N. Houstis, Z.Ma, S. Markus, J.R. Rice, N.H. Wang, S. Weerawarana, The SoftLab experience: Building virtual laboratories for computational science, CSD-TR-95-041, Dept. of Computer Sciences, Purdue Univ., 1995.
12. T.F. Chan, R. Glowinski, J. Periaux and D. Widlund, *Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, SIAM Pubs (1990).
13. N. Chrisochoides, E.N. Houstis, and J.R. Rice, Mapping algorithms and software environment for data parallel PDE iterative solvers, *Journal of Parallel and Distributed Computing*, 21, 75-95 (1994).
14. Component Integration Laboratories, *OpenDoc: The New Shape of Software*, Sunnyvale, Calif. (1994).
15. M.A. Cornea-Hasegan, C. Costian, D.C. Marinescu, I. Martin, and J.R. Rice, Towards problem solving environments for high performance computing, *High Performance Computing '94*, National Supercomputer Research Center, Singapore (1994), 354-366.
16. T. Drashansky, A. Joshi and J.R. Rice, *SciAgents- An Agent Based Environment for Distributed, Cooperative Scientific Computing*, CSD-TR-95-029, Department of Computer Sciences, Purdue University, 1995.
17. T. Drashansky, S. Weerawarana, A. Joshi, R. Weerasinghe, E.N. Houstis, Software architecture of ubiquitous scientific computing environments for mobile platforms, CSD-TR-95-032, Dept. of Computer Sciences, Purdue Univ., 1995.

18. C. Farhat and L. Crivelli and F.-X. Roux, Extending substructure based iterative solvers to multiple load and repeated analyses, *Comput. Methods Appl. Mech. Engrg.* **117** (1994), 195-209.
19. T. Finin, Y. Labrou, and J. Mayfield, KQML as an agent communication language, in J. M. Bradshaw Ed. *Software Agents*, MIT Press, 1997, pp. 291-316.
20. S. Fleeter, E. N. Houstis, J. R. Rice, C. Zhou, GasTurbnLab Design, Technical Report, Department of Computer Science, Purdue University, CSD-TR #99-003, January 1999.
21. S. Fleeter, E. N. Houstis, J. R. Rice, C. Zhou, Gas Turbine Engine Compressor – Combustor Dynamics Simulation Design, Technical Report, Department of Computer Science, Purdue University, CSD-TR #99-006, February 1999.
22. E. Gallopoulos, E.N. Houstis, and J.R. Rice, Future research directions in problem solving environments for computational science, CSD-TR-92-032, Dept. of Computer Sciences, Purdue Univ., 1992.
23. E. Gallopoulos, E.N. Houstis, and J.R. Rice, Computer as thinker/doer: Problem solving environments for computational science. *IEEE Comp. Sci. Engr.*, **1** (1994), 11–23.
24. M. R. Genesereth, An agent-Based Framework for Interoperability, in J. M. Bradshaw Ed. *Software Agents*, MIT Press, 1997, pp.317-345.
25. R. Glowinski, G. Golub, G. Meurant, and J. Periaux, *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, SIAM Pubs. (1988).
26. S.S. Grimajl, The first ICASE/LARC industry roundtable: Session proceedings, ICASE Interim Report 26, ICASE, NASA Langley Research Center, Hampton, VA, 1995.
27. C.M. Hoffmann, E.N. Houstis, J.R. Rice, A.C. Catlin, M. Gaitatzes, S. Weerawarana, N-H L. Wang, C.G. Takoudis, and D.G. Taylor, SoftLab – A virtual laboratory for computational science. *Math Comp. Simulation* **36** (1994), 479– 491.
28. E.N. Houstis, J.R. Rice, and R. Vichnevetsky (editors), *Intelligent Mathematical Software Systems*, North Holland, Amsterdam (1990), 323 pages.
29. E.N. Houstis and J.R. Rice, Parallel ELLPACK: A development and problem solving environment for high performance computing machines. In *Programming Environments for High-Level Scientific Problem Solving* (P. Gaffney and E. Houstis, eds.), North-Holland, Amsterdam (1992), 229–241.
30. E.N. Houstis and J.R. Rice, The architecture of PDE solving systems. In *Computer Methods for Partial Differential Equations VII* (R. Vichnevetsky, ed.), IMACS, New Brunswick, NJ (1992), 363–370.
31. E.N. Houstis, J.R. Rice, and S. Weerawarana, A software platform for integrating symbolic computation with a PDE solving environment. *Proc. 14th IMACS World Congress*, IMACS **1**, (1994), 482–485.
32. E.N. Houstis, J.R. Rice, and S. Weerawarana, An open structure for PDE solving systems. *Proc. 14th IMACS World Congress*, **3** (1994), 1296–1299.
33. Industrial Research Institute, *Proceedings: Roundtable meeting on reducing R&D cycle time*, Industrial Research Inst., Washington DC, (1992).
34. W.R. Johnson, Jr., Anything, Anytime, Anywhere: The future of networking, in *Technology 2001: The future of Computing and Communications*, D. Leebaert, editor, MIT Press, Cambridge, MA, (1992).

35. A. Joshi, S. Weerawarana, E.N. Houstis, J.R. Rice, N. Ramakrishnan, On using computational intelligence to support problem solving environments for scientific computing, CSD-TR-95-040, Dept. of Computer Sciences, Purdue Univ., 1995.
36. D.E. Keyes, T. F. Chan, G. Meurant, J. S. Scroggs, and K.G. Voigt, *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, SIAM Pubs. (1992).
37. S.B. Kim, A. Hadjidimos, E.N. Houstis, and John R. Rice, The performance of parallel stationary iterative methods for distributed memory machines, *Proceedings of the Intel Supercomputer Users Group*, (1994), 169-173.
38. S.B. Kim, S. Markus, N.E. Houstis, E.N. Houstis, A.C. Catlin, P. Wu, Parallel methodologies for “legacy” scientific software, *Proc. Intel Supercomputer Users Group*, (1995).
39. J. Ledeborg and K. Uncapher, *Towards a national collaboratory, Report of an Invitational Workshop at The Rockefeller University*, March, 1989.
40. D.C. Marinescu, J.R. Rice, B. Waltsburger, C.E. Houstis, T. Kung, and H. Waldschmidt, Distributed supercomputing. In *Future Trends '90*, IEEE Press (1990), 381–387.
41. E. Mascarenhas, V. Rego, Ariadne: Architecture of a portable threads system supporting mobile processes, CSD-TR-95-017, Department of Computer Sciences, Purdue University.
42. E. Mascarenhas, F. Knop , V. Rego, ParaSol: A multiThreaded System for Parallel Simulation based on mobile threads, submitted for publication, 1995.
43. H.S. McFaddin, *An Object Based Problem Solving Environment for Composite Partial Differential Equations*, Ph.D. thesis, Department of Computer Sciences, Purdue University, 1992.
44. H.S. McFaddin and J.R. Rice, Collaborating PDE solvers. *Applied Numerical Mathematics*, **10**, (1992), 279–295.
45. H.S. McFaddin and J.R. Rice, RELAX: A platform for software relaxation. In *Expert Systems for Scientific Computing*, (Houstis, Rice, and Vichnevetsky, eds.), North-Holland, Amsterdam (1992), 175–194.
46. M. Mu and J.R. Rice, Modeling with collaborating PDE solvers – Theory and practice. *Contemporary Mathematics*, **180**, (1994), 427–438.
47. Mo Mu and John R. Rice, Collaborating PDE solvers with interface relaxation, *submitted for publication*.
48. M. Mu and J.R. Rice, Preconditioning for domain decomposition through functional approximation, *SIAM J. Sci. Comp.*, **15** (1994), 1452-1466.
49. Object Management Group, Common Facilities Architecture Rev. 4.0, *OMG Document No. 95.1.2*, Framingham, Mass., (1995).
50. A. Quarteroni, F. Pasquarelli, and A. Valli, Heterogeneous domain decompositions: Principles, algorithms, applications, in *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations (D. Keyes et. al, eds.)*, SIAM Pubs. (1992), 129-150.
51. J.R. Rice and H.D. Schwetman, Interface issues in a software parts technology. In *Reusability in Programming*, (Biggerstaff, ed.), ITT Technology (1983), 129–137. Reprinted in *Software Reusability* P. Freeman, ed.), IEEE Tutorial, Computer Society Press (1987), 96–104. Revised version in *Software Reusability* (T.J. Biggerstaff and A.J. Perlis, eds.), ACM Press (1989), 125–139.

52. Taylor and T.G. Hughes, *Finite element programming of the Navier-Stokes equations*, Pineridge Press, Swansea, U.K., (1981).
53. William F. Tidd, James R. Rinderle, and Andrew Witkin, Design refinement via interactive manipulation of design parameters and behaviors, *Proceedings of the 4th Design Theory and Methodology Conference*, 1992.
54. P. Tsompanopoulou, L. Boloni, D. Marinescu, and J. Rice, The design of software agents for a network of PDE solvers. Proc. Agents' 99: Third Intl. Conf. Autonomous Agents, ACM Press, 1999.
55. J.T. Vesey, Speed-to-Market distinguishes the new competitors, *Research Tech. Mgmt.*, **34** (1991), 33-38.
56. R. G. Voigt, Requirements for multidisciplinary design of aerospace vehicles on high performance computers, ICASE Report No. 89-70, Sept. 1989, 9 pages.
57. Way cool science on the next Internet, *Bussiness week*, May 1, 1995, 25.
58. S. Weerawarana and P.S. Wang, A portable code generator for CRAY FORTRAN, ACM Trans. Math. Soft., **18** (1992), 241-255.
59. S. Weerawarana, E.N. Houstis, and J.R. Rice, An interactive symbolic-numeric interface to parallel ELLPACK for building general PDE solvers. In *Symbolic and Numerical Computation for Artificial Intellgence*, (Donald, Kapur and Mundy, eds.), Academic Press, (1992), 303–321.
60. S. Weerawarana, E.N. Houstis, J.R. Rice, A.C. Catlin, C.L. Crabill, C.C. Chui, S. Markus, PDELab: An object-oriented framework for building problem solving environments for PDE based applications, *Proc. 2nd Object-Oriented Numerics Conf.*, (A. Vermeulen, ed.), RogueWare Software, Corvallis, OR (1994), 79–92.
61. S. Weerawarana, *Problem Solving Environments for Partial Differential Equation Based Systems*, Ph.D. Thesis, Department of Computer Sciences, Purdue University, 1994.
62. S. Weerawarana, E. Houstis, J.R. Rice, A.C. Catlin, M.G. Gaitatzes, C.L. Crabill, S. Markus, and T.T. Drashansky, Towards a kernel for building PSEs, to appear in *Problem Solving Environments for Computational Science* (E. Houstis, S. Gallopoulos, J. Rice and R. Brambley, eds.), IEEE Press, Los Alamitos, CA, 2000.
63. P. Wu, E.N. Houstis, and J.R. Rice, EPPOD: A parallel problem solving environment for the electronic prototyping of physical objects design, *Proc. DAGS '94 Symposium*, (F. Makedon, ed.), Dartmouth Inst. Adv. Grad. Studies, Dartmouth, NH (1994), 135–151.
64. P. Wu and Elias N. Houstis, A parallel mesh generation and decomposition methodology, *Proceedings of Mesh Generation Conference*, Albuquerque, Oct. 1994.
65. P. Wu, Parallel Shape Optimization, *Proc. Intl. Conf. on Parallel Algorithms (ICPA'95)*, Wuhan-China, 1995.
66. D.M. Young, The search for “high-order” parallelism for iterative sparse linear system solvers, *Parallel Supercomputing: Methods, Algorithms and Applications*, G.F. Carey, ed., 1989, 89-106.
67. D.M. Young and B.R. Vona, On the use of rational iterative methods for solving large linear systems, *Appl. Numer. Math.*, **10**, 1992, 261-278.
68. Sun Microsystems, The Network is the Computer, Trademark.
69. The Grasshopper Agent Platform, IKV++ GmbH, Kurfurstendamm 173-174, D-10707 Berlin, Germany. <http://www.ikv.de>.

70. IRIS Explorer Toolkit, IRIS Explorer Center (North America), 1400 Opus Place, Suite 200, Downers Grove, IL 60515-5702, USA. <http://www.nag.com/IEC>.
71. TrueGrid, XYZ Scientific Applications Inc., USA. <http://www.truegrid.com>.

---

## APPENDIX A

---

### SOFTWARE AVAILABILITY

Software availability on the hardware platforms under consideration imposes certain restrictions on the design.

SOFTWARE	SOLARIS X86	SGI	WIN32	OTHER
KIVA-3V	X	X	X	
ALE3D	X	X		
PELLPACK	X	X		SunOS5
IRIS Explorer		X	X	Linux
TrueGrid	X	X	X	SunOS5
Grasshopper	X	X	X	SunOS5, Linux

Hence the user interface for the GasTurbnLab PSE will run on Win32 platforms and the computational grid will initially consist of SGIs and a Solaris x86 machine cluster.

---

**APPENDIX B**

---

**INTERFACE RELAXATION AND INTERPOLATION IMPLEMENTATION**

**FUNCTIONAL DESCRIPTION**

<b>Function Name</b>	<b>Purpose</b>	<b>Input</b>
ReadInterfacePoints	Reads interface points from a file for both solvers (left and right).	file format :: x y z
ReadInterpolationPoints	Reads interpolation points from a file for each interface point.	file format :: x y z
ReadInterfaceValues	Reads the values of a function and its derivatives for each interface point.	file format :: u ux uy uz
ReadInterpolationValues	Reads the values of a function and its derivatives for each interpolation point.	file format :: u ux uy uz
Interpolate	Calls the deBoor interpolation functions to create and evaluate the polynomials for each interface point using the interpolation points and the values of the appropriate functions.	
ApplyRelaxation	Implementation of the relaxation method in the draft of "Initial Interface Relaxation Trial for ALE3D".	

**INTERPOLATION IMPLEMENTATION NOTES**

When eight points are used to construct the interpolant, the MVP library gives polynomials with degree up to seven for all three variables (x, y and z). However, most of the coefficients of the monomials are almost zero ( $e-14$ ) and only the coefficients of x, y, z and the constant term are large enough. Although this is dependent on the data used for the tests, the number of points to be used for the interpolation need to be decided.

If u, ux, uy and uz are provided from the solvers, it is not clear if the mediator should find interpolants for the above functions or if it will differentiate the interpolant of u to find the interpolants of ux, uy and uz.

## INTERFACE RELAXATION CODE

```
voidApplyRelaxation(interf)
  Interface **interf;
{
  InterfacePointPtr ipcurrent;
  double u, ux, uy, uz;

  /*
   * update the values of the left solver points
   */
  ipcurrent = (*interf)->ipl;
  while (ipcurrent != NULL) {
    ipcurrent->u = (ipcurrent->u + ipcurrent->v)/2. +
      (*interf)->relaxparam[0] *
      (ipcurrent->ux*ipcurrent->nx - ipcurrent->vx*ipcurrent->nx +
       ipcurrent->uy*ipcurrent->ny - ipcurrent->vy*ipcurrent->ny +
       ipcurrent->uz*ipcurrent->nz - ipcurrent->vz*ipcurrent->nz);
    fprintf(stderr, "new value at point ( %lf %lf %lf ) = %lf\n",
             ipcurrent->x, ipcurrent->y, ipcurrent->z, ipcurrent->u);
    ipcurrent = ipcurrent->next;
  }
  (*interf)->iplcurrent = ipcurrent;

  /*
   * update the values of the right solver points
   */
  ipcurrent = (*interf)->ipr;
  while (ipcurrent != NULL) {
    ipcurrent->u = (ipcurrent->u + ipcurrent->v)/2. +
      (*interf)->relaxparam[0] *
      (ipcurrent->ux*ipcurrent->nx - ipcurrent->vx*ipcurrent->nx +
       ipcurrent->uy*ipcurrent->ny - ipcurrent->vy*ipcurrent->ny +
       ipcurrent->uz*ipcurrent->nz - ipcurrent->vz*ipcurrent->nz);
    fprintf(stderr, "new value at point ( %lf %lf %lf ) = %lf\n",
             ipcurrent->x, ipcurrent->y, ipcurrent->z, ipcurrent->u);
    ipcurrent = ipcurrent->next;
  }
  (*interf)->iprcurrent = ipcurrent;
}
}
```

## INTERPOLATION CODE

```
void Interpolate(interf, sol)
  Interface **interf;
  SolverType sol;
{
  InterfacePointPtr ipcurrent;
  FILE *fp;
  double *xyz, *bounds;
  double *x;
  MVPOLY *poly;
  long int ier, incx;

  if (sol == LEFT) {
    ipcurrent = (*interf)->ipl;
  } else {
    ipcurrent = (*interf)->ipr;
  }

  if ((fp = fopen("POLYNOMIALS","a")) == NULL) {
    fprintf(stderr,"interface: can't open POLYNOMIALS file \n");
    return ;
  }

  x = (double *) calloc(3, sizeof(double));

  /*
   * for each interface point compute the interpolant function and its
   * value using its interpolation points
   */
  while( ipcurrent != NULL) {
    xyz = ipcurrent->xyz;
    bounds = ipcurrent->bounds;
    /*
     * create the polynomial for the current point
     */
    ier = mvpint(3, 1, ipcurrent->nip, xyz, 4 , 1, bounds, &poly);
    if (ier != 0 ) {
      fprintf(stderr, "Error:: creation of multivariant polynomial didn't
end properly.\n");
      return;
    }
    /*
     * print the polynomial in a file to check the coefficients
     */
    ier = mvpwrt(fp, poly);
    if (ier != 0 ) {
      fprintf(stderr, "Error:: printing of multivariant polynomial didn't
end properly.\n");
      return;
    }
  }
}
```

```

/*
 * evaluate the polynomial on the interface point
 */

x[0] = ipcurrent->x;
x[1] = ipcurrent->y;
x[2] = ipcurrent->z;
ier = mvpval(x, 1, poly, &(ipcurrent->v));
if (ier != 0 ) {
    fprintf(stderr, "Error:: evaluation of multivariant polynomial
didn't end properly.\n");
    return;
}
fprintf(stderr, "Evaluation of poly at (%lf, %lf, %lf ) == %lf
\n", x[0], x[1], x[2], ipcurrent->v);

/*
 * deallocate the space of the polynomial
 */

ier = mvfree(&poly);
if (ier != 0 ) {
    fprintf(stderr, "Error:: deallocation of multivariant polynomial
didn't end properly.\n");
    return;
}

/*
 * move to the next point
 */

ipcurrent = ipcurrent->next;
}

/*
 * some necessary things
 */

x = NULL;
if (sol == LEFT) {
    (*interf)->iplcurrent = ipcurrent;
} else {
    (*interf)->iprcurrent = ipcurrent;
}

fclose(fp);
}

```

---

## APPENDIX C

---

### IMPLEMENTATION OF A JAVA WRAPPER AROUND A LEGACY LINEAR SOLVER

JAVA

DRIVER

CODE

```
/*
 * Java driver for the itpack fortran library.
 * Calls the native C method, ItpackWrapper, to access the fortran routines.
 * All matrices are stored in column-major order in arrays.
 */

public class ItpackDriver {

    /* Constants */

    public static final int JCG = 1;
    public static final int JSI = 2;
    public static final int SOR = 3;

    public static final float ZETA = (float) 0.000005;
    public static final float CME = (float) 0.0;
    public static final float SME = (float) -1.0;
    public static final float FF = (float) 0.75;
    public static final float OMEGA = (float) 1.0;
    public static final float SPECR = (float) 0.0;
    public static final float BETAB = (float) 0.25;
    public static final int ITMAX = 100;
    public static final int ISYM = 0;
    public static final int LlTIME = 0;
    public static final int IADAPT = 1;
    public static final int ICASE = 1;
    public static final int IDGTS = 0;

    /* Variable Members */

    int module;
    int iparm[];
    float rparm[];
    float rlunkn[];

    /* Static code block to load the native libraries */

    static {
        System.loadLibrary ("ItpackWrapper");
    }
}
```

```

/* Native methods */

public native
    int ItpackWrapper(int module, int ilneqn, int ilmneq,
        int ilncoe, int[] ilidco, float[] rlcoef,
        float[] rlbbbb, float[] rlunkn, int[] iparm,
        float[] rparm, int ier);

/* Constructor */

public ItpackDriver(LinearSystemParser parser, int itpackModule)
{
    int ier = 0;
    int ilneqn, ilmneq, ilncoe;
    int ilidco[];
    float rlcoef[];
    float rlbbbb[];

    // parse XML format input file
    ilneqn = parser.getNeqn();
    Indexer ndxr = new Indexer(ilneqn);
    rlcoef = parser.getCoef();
    ilidco = parser.getIdcoef();
    rlbbbb = parser.getRhs();
    rlunkn = new float[ilneqn];
    iparm = new int[12];
    rparm = new float[12];
    ilmneq = ilneqn;
    ilncoe = ilneqn;
    module = itpackModule;

    setParams(ITMAX, 5, -2, ISYM, IADAPT, ICASE, IDGTS, L1TIME, ZETA,
        CME, SME, FF, OMEGA, SPECR, BETAB);

    // call the itpack c wrapper (native) method
    int ret = ItpackWrapper(module, ilneqn, ilmneq, ilncoe, ilidco,
        rlcoef, rlbbbb, rlunkn, iparm, rparm, ier);
    if (ret == 0)
        System.out.println("itpack c wrapper successfully executed");
    else {
        System.out.println("itpack c wrapper error = " + ret);
        return;
    }
    System.out.println("ier = " + ier);
    System.out.println("iterations = " + iparm[0]);
}

```

```

/* Set parameters */
public void setParams(int itmax, int outputLevel, int nb, int isym,
    int iadapt, int icase, int idgts, int lltime,
    float zeta, float cme, float sme, float ff,
    float omega, float specr, float betab)
{
    iparm[0] = itmax;
    iparm[1] = outputLevel;
    iparm[2] = 0;
    iparm[3] = 7;
    iparm[4] = isym;
    iparm[5] = iadapt;
    iparm[6] = icase;
    iparm[7] = 0;
    iparm[8] = nb;
    iparm[9] = 0;
    iparm[10] = lltime;
    iparm[11] = idgts;
    rparam[0] = zeta;
    rparam[1] = cme;
    rparam[2] = sme;
    rparam[3] = ff;
    rparam[4] = omega;
    rparam[5] = specr;
    rparam[6] = betab;
    rparam[8] = (float) 0.0;
    rparam[9] = (float) 0.0;
    rparam[10] = (float) 0.0;
    rparam[11] = (float) 0.0;
}

/* Accessor Methods */
public float[] getSoln() { return rlunkn; }

/* Main driver */
public static void main (String args[])
{
    // check arguments
    if (args.length == 0) {
        System.err.println("Usage: java LinearSystemParser input_file");
        System.exit(1);
    }

    // read in the linear system
    LinearSystemParser parser = new LinearSystemParser(args[0]);
    ItpackDriver driver = new ItpackDriver(parser, itpackModule);

    // output solution vector
    System.out.println("solution vector :");
    float[] soln = driver.getSoln();
    for (int i = 0; i < parser.getNeqn(); ++i)
        System.out.println(" " + soln[i]);
}
}

```

## C WRAPPER CODE

```
/*
 * C wrapper for the itpack (pellpack version) modules
 * -----
 *
 * To solve:  a*x = b
 *   where matrix a is stored in the ellpack format.
 *
 * module : itpack module to be called [ 1=jcg, 2=jsi, 3=sor, 4=ssorcg,
 *   5=ssorsi, 6=rscg, 7=rssi ]
 * ilneqn : no. of eqns
 * ilmneq : row dimension of rlcoe & ilidco in ellpack program
 * ilncoe : max no. of nonzeros per eqn
 * ilidco : coef identity (column numbers) matrix (dim: ilmneq x ilncoe)
 * rlcoef : nonzero of coef matrix (dim: ilmneq x ilncoe)
 * rlbbbb : rhs (dim: ilneqn)
 * rlunkn : solution array (dim: ilneqn)
 * iwksp  : integer workspace array (dim: ilneqn)
 * ilkwrk : 2 * ilmneq
 * rlwork : real workspace array (dim: depends on the itpack module)
 * iparm  : integer parameters (dim: 12) [ itmax/iter, level, iredset,
 *   iloutp, isym, iadapt, icase, nwksp, nblack, iremov, itime,
 *   idgts ]
 * rparm  : real parameters (dim: 12) [zeta, cme, sme, ff, omega, specr,
 *   betab, tol, timel, time2, digit1, digit2 ]
 * ier    : output error condition level
 *
 * -----
 * Return Codes:
 * 0 = successful C wrapper execution
 * 1 = unsuccessful for unspecified reason
 * 2 = itmax (iparm[0]) set to zero
 * 3 = nblack (iparm[8]) is invalid for rscg/rssi modules
 */

#include "ItpackDriver.h"

#define ind(i,j,n)  ((j*n)+i)

JNIEXPORT jint JNICALL
Java_ItpackDriver_ItpackWrapper(JNIEnv * jenv, jobject this, jint module,
    jint ilneqn, jint ilmneq, jint ilncoe,
    jintArray ilidco_j, jfloatArray rlcoef_j,
    jfloatArray rlbbbb_j, jfloatArray rlunkn_j,
    jintArray iparm_j, jfloatArray rparm_j,
    jint ier)
{
    int* iwksp;
    float* rlwork;
    int i, j, ilkwrk, dim, itmax=100, nb, l;
    jint *ilidco, *iparm;
    jfloat *rlcoef, *rlbbbb, *rlunkn, *rparm;

    printf("ItpackWrapper function\n");
}
```

```

/* Handle Arrays */
ilidco = (*jenv)->GetIntArrayElements(jenv, ilidco_j, 0);
rlcoef = (*jenv)->GetFloatArrayElements(jenv, rlcoef_j, 0);
rlbbbb = (*jenv)->GetFloatArrayElements(jenv, rlbbbb_j, 0);
rlunkn = (*jenv)->GetFloatArrayElements(jenv, rlunkn_j, 0);
iparm = (*jenv)->GetIntArrayElements(jenv, iparm_j, 0);
rparm = (*jenv)->GetFloatArrayElements(jenv, rparm_j, 0);

/* Allocate workspaces */
iwksp = (int *) calloc(ilneqn, sizeof(int));
ilkwrk = 2 * ilmneq;
itmax = iparm[0];
if (!itmax) return 2;
l = iparm[4] ? 4 : 2;
nb = iparm[8];
if ((module >= 6) && (nb < 0)) return 3;
switch (module) {
case 1: dim = 5*ilneqn + l*itmax; break;
case 2: dim = 3*ilneqn; break;
case 3: dim = 2*ilneqn; break;
case 4: dim = 7*ilneqn + l*itmax; break;
case 5: dim = 6*ilneqn; break;
case 6: dim = 2*ilneqn + 3*nb + l*itmax; break;
case 7: dim = 2*ilneqn + nb; break;
}
iparm[7] = dim;
rlwork = (float *) calloc(dim, sizeof(float));

/* Call the Itpack module fortran routines */
switch (module) {
case 1: q5i2m1_(&ilneqn, &ilmneq, &ilncoe, ilidco, rlcoef, rlbbbb,
               rlunkn, iwksk, &ilkwrk, rlwork, iparm, rparm, &ier);
        break;
case 2: q5i3m1_(&ilneqn, &ilmneq, &ilncoe, ilidco, rlcoef, rlbbbb,
               rlunkn, iwksk, &ilkwrk, rlwork, iparm, rparm, &ier);
        break;
case 3: q5ilm1_(&ilneqn, &ilmneq, &ilncoe, ilidco, rlcoef, rlbbbb,
               rlunkn, iwksk, &ilkwrk, rlwork, iparm, rparm, &ier);
        break;
}

/* Release memory allocated to the primitive arrays */
(*jenv)->ReleaseIntArrayElements(jenv, iparm_j, iparm, 0);
(*jenv)->ReleaseFloatArrayElements(jenv, rlbbbb_j, rlbbbb, 0);
(*jenv)->ReleaseFloatArrayElements(jenv, rlunkn_j, rlunkn, 0);
(*jenv)->ReleaseFloatArrayElements(jenv, rparm_j, rparm, 0);
(*jenv)->ReleaseIntArrayElements(jenv, ilidco_j, ilidco, 0);
(*jenv)->ReleaseFloatArrayElements(jenv, rlcoef_j, rlcoef, 0);

return (jint) 0;
}

```

---

## APPENDIX D

---

### SAMPLE XML-FORMAT USED IN THE DATABASES

```
<?xml version="1.0"?>

<LinearSystem name="test05">
  <neqn> 10 </neqn>
  <sparseMatrix name="coef" rows="10" cols="6" type="float">
    <file line="1">
      file://nuwan.cs.purdue.edu/DB/LinearSystems/05/mat.data
    </file>
  </sparseMatrix>
  <sparseMatrix name="idcoef" rows="10" cols="6" type="int">
    <file line="12">
      file://nuwan.cs.purdue.edu/DB/LinearSystems/05/mat.data
    </file>
  </sparseMatrix>
  <array name="rhs" size="10" type="float">
    <r> 0.526460266E+02 </r>
    <r> 0.467939034E+02 </r>
    <r> 0.751092529E+02 </r>
    <r> 0.582886505E+02 </r>
    <r> 0.567966385E+02 </r>
    <r> 0.653487778E+02 </r>
    <r> 0.682942581E+02 </r>
    <r> 0.380436096E+02 </r>
    <r> 0.418298378E+02 </r>
    <r> 0.470281410E+02 </r>
  </array>
</LinearSystem>
```

---

**APPENDIX E**

---

**RESOURCE CHARACTERISTIC EVENTS**

**HOST EVENTS TABLE**

<b>Event ID</b>	<b>Event Description</b>
221	HL1: Load average above 2.0
222	HL2: Load average above alert level
220	HL0: Load average returned to acceptable
231	HM1: Memory usage above 95%
232	HM2: Memory usage above alert level
230	HM0: Memory usage returned to acceptable
241	HP1: Number of running processes above alert level
240	HP0: Number of running processes returned to acceptable
251	HS1: Host shutdown alert
261	HN1: Special alert
260	HN0: Host performance level acceptable

**NETWORK PERFORMANCE EVENTS TABLE**

<b>Event ID</b>	<b>Event Description</b>
41001	NB001: Average hops to base server above alert level
41000	NB000: Average hops to base server at acceptable level
41201	NB201: Ping response time to base server above alert level
41200	NB200: Ping response time to base server at acceptable level
41301	NB301: Host disconnecting from network
41300	NB300: Host network connected
42001 – 42099	NU001 – NU099: Neighbor host 1-99 unreachable
43001 – 43099	NR001 – NR099: Neighbor host 1-99 reachable
44001 - 44099	NS001 – NS099: Response time to neighbor host 1-99 above alert level
45001 - 45099	NF001 – NF099: Response time to neighbor host 1-99 below alert level