

•
•
•
•
•
•

GASTURBNLAB:
A PROBLEM-SOLVING ENVIRONMENT FOR
GAS TURBINE ENGINE DESIGN ON A
NETWORK OF NON-HOMOGENEOUS
MACHINES

**E. N. Houstis, A. C. Catlin, G. Balakrishnan,
D. Gottfried, K. Su, P. Tsompanopoulou and J. R. Rice**

ABSTRACT

Gas turbine engines are very complex (20-40,000 parts) and have extreme operating conditions (10-50,000 RPM, 1000-1500 °F, pressures of 20-50 atmospheres, and 5-10g loads). The important physical phenomena take place on scales from microns to meters. A complete and accurate dynamic 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. Designing such complex systems will use fast, accurate simulations of computational models from multiple engineering disciplines, along with sophisticated optimization techniques to help guide the design process. In this paper we present a general framework for building a multidisciplinary problem solving environment (MPSE) which operates on a network of non-homogeneous computer resources. We describe the implementation of GasTurbnLab using this framework, and show how the GasTurbnLab MPSE can be used for the design of efficient gas turbine engines.

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 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 (behavior of the gases in the piston-cylinder assemblies), mechanics (kinematics and dynamic behaviors of pistons, links, cranks, etc.), structures (stresses and strains on the parts) and geometry (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. 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 implement 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 a multi-physics application related to the simulation of gas turbine engines.

This document is organized as follows Section 2 defines the concept of the multidisciplinary problem solving environment and reviews the associated research issues. Section 3 describes the gas turbine engine as a multi-physics application. Section 4 presents the design and application infrastructure of the agent-based MPSE framework. Section 5 describes the implementation of the GasTurbnLab MPSE using this framework. In Section 6, we discuss results of prototype GasTurbnLab engine simulations, and further simulation studies are outlined in Section 7. We conclude our discussion in Section 8 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 GasTurbnLab.

SECTION 2: MPSE - DEFINITIONS AND RESEARCH ISSUES

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

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 (1) a PSE kernel for building scientific PSEs [62], (2) a knowledge based framework to address computational intelligence issues for PDE based PSEs [28][35], (3) infrastructure for solving PDEs [29][30][31][59][61], (4) parallel PDE methodologies [13][38][40][63][64][65] and (5) 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 is applicable to a wide variety of practical problems, allows for software reuse in order to achieve lower costs and high quality, and 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.

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 the 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., everywhere by everyone. 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 an MPSE will receive only the user interface at his location. Thus, the MPSE server will export an agent to the user's machine 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 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 the collaboration is built on standard network services, and all can be managed without involving the user, 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 large 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 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 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 a single scientist to create 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 partitioning, 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. In the following section, we propose a software framework for MPSEs supporting PDE based applications and describe its implementation 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 validate the results. 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 to as GasTurbnLab, to study complex physical phenomena such as stall, surge and turbine blade fatigue. Figure 2 presents an abstraction of a multi-physics application and the corresponding functional requirements. The hardware infrastructure for these simulations and the implementation of MPSE consists of a computational grid involving three 16 i86 Intel PC clusters running Solaris, an SP-2, and an SGI Origin 2000 with 32 CPUs. In this study we utilize the *Grasshopper Agent Platform* [69] which is MASIF (Mobile Agent System Interoperability Facilities Specification) standard.

SECTION 4: FRAMEWORK FOR AN AGENT-BASED MPSE

In this section we describe the design of a general MPSE framework that can be used to simulate complex multi-physics phenomena governed by PDE models. A network of distributed machines is the hardware infrastructure. Since PDE simulations are often defined on geometric domains, natural or artificial geometric boundaries can be used to split the problem and the underlying simulation into

many smaller sub-problems. Each sub-problem can then be solved independently, with mediator interactions along the boundaries for interface relaxation. Thus, the MPSE framework for composite PDE simulations is based on domain decomposition with geometric objects, using a network of PDE solvers and interface mediators executing on a network of distributed machines.

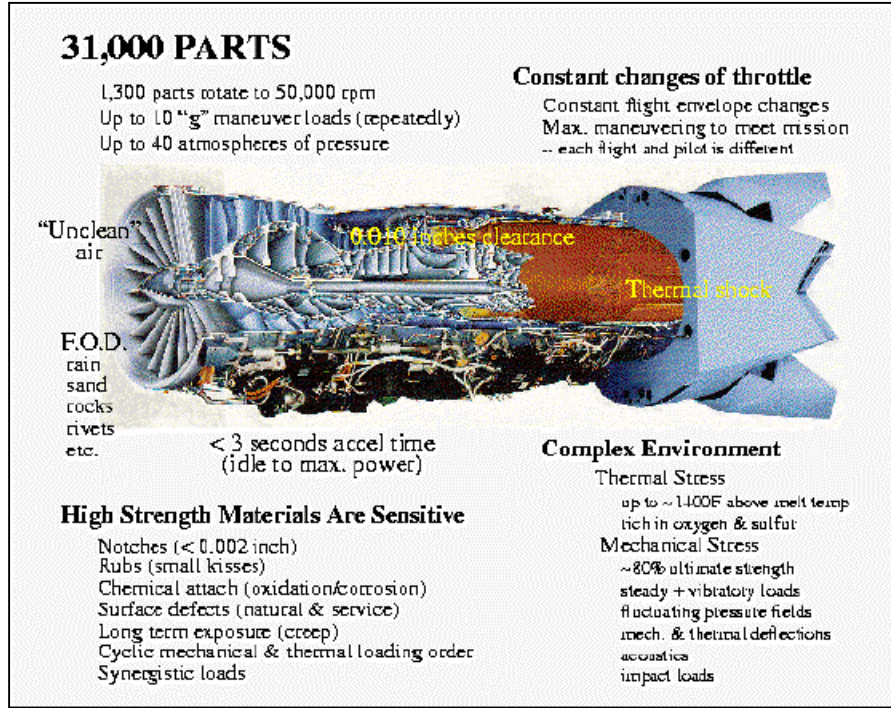


Figure 1: View of a gas turbine showing some of its detail, some of its operational characteristics and the engineering methodologies involved in its design, simulation and construction.

FUNCTIONAL SPECIFICATIONS OF THE MPSE

The design of the MPSE framework is driven by the underlying geometric modularity of the problem. The geometry is assumed to have a root node for the target physical object (domain), and the user is allowed to subdivide it in multiple ways. The resulting network of geometric objects defines a network of PDE models, where each object (subdomain) is modeled by a PDE. Each subdomain has some neighbors and possibly some fixed boundaries. If each neighboring connection is represented by an arrow, we get an abstraction of a network of PDE models. Since the PDEs on each subdomain are usually not the same, this represents a composite PDE problem. The MPSE framework maps the network of PDE models resulting from a user-specified partitioning onto a set of computational solvers running on a network of machines. This resource allocation will be done in an optimal manner to minimize the communication overhead between computational agents of neighboring subdomains.

Under the assumption that any single PDE problem of the composite problem can be solved exactly, the interface relaxation technique will be used to solve the composite PDE problem. The interface relaxation algorithm is based on the iteration shown in Figure 3. This MPSE supports user specification of a set of geometric objects that partition a composite physical domain, with a corresponding network of PDE solvers that collaborate via mediation to find a solution for the composite problem.

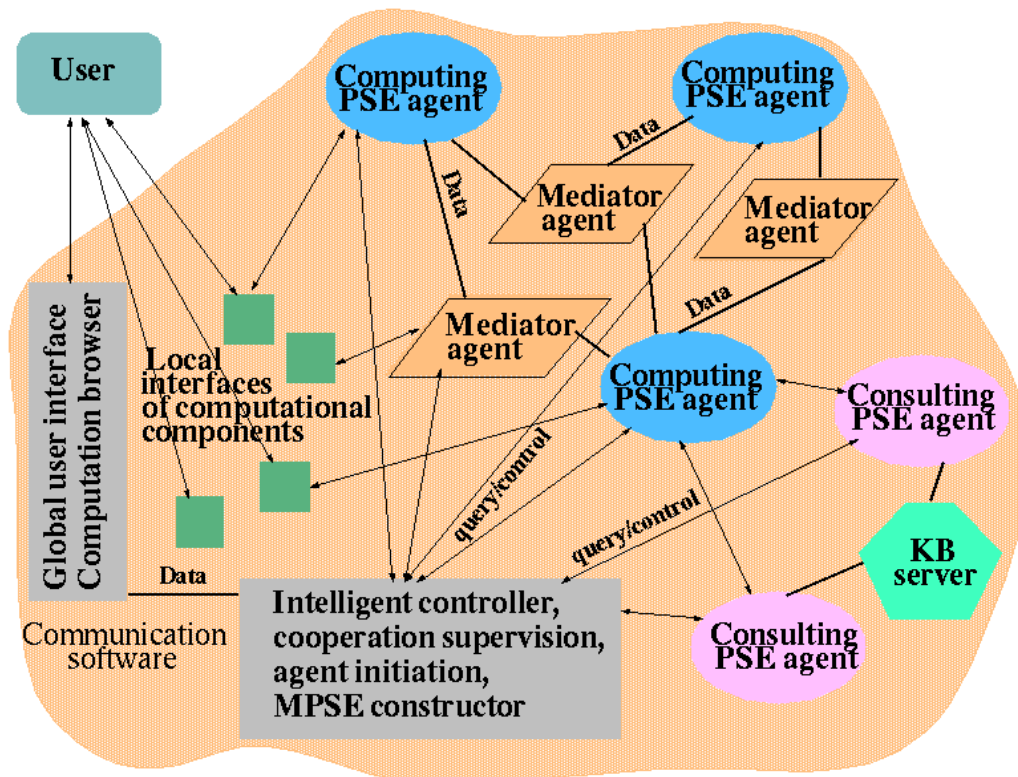


Figure 2: Functional view of a multidisciplinary PSE. The computations and major data exchanges are concentrated in the network of solver and mediator agents. Users interact with the system through the global and local GUIs, which send queries and receive replies from the various agents.

The MPSE framework relies on an agent-based infrastructure. All system components, including the user interface, the “legacy” computational PDE solvers and the mediators, operate as agents. These agents are implemented using the Grasshopper Agent Platform [69]. A brief discussion of Grasshopper is given first, since a description of its functionality is essential for describing the MPSE agent-based framework.

THE GRASSHOPPER AGENT SYSTEM

Grasshopper is an agent development platform that supports distributed agent-based applications. The platform provides a base for communication services, mobile computing power and dynamic information retrieval. It is essentially a mobile agent platform that is built on top of a distributed processing environment, integrating the traditional client-server paradigm and mobile agent technology. The primary feature of the Grasshopper platform is its location independent computing, driven by the ability to move agents between different systems. It is a powerful environment that facilitates the creation of agents, transparently locating them and controlling their execution. These agent-based applications are interoperable with other agent systems that are MASIF (Mobile Agent System Interoperability Facilities) compliant. The Grasshopper distributed agent environment is composed of region, agencies and agents. At the top of the hierarchy is the *region* that manages the distributed components in the Grasshopper environment. Agencies and agents are associated with a

particular region. Each region has a registry that maintains information about all components associated with it. An *agency* is the runtime environment for mobile and stationary agents, providing the functionality to support agent execution. The agency is responsible for a number of services, including (1) communication services for all remote interactions that take place between Grasshopper components, their movements and transport, (2) registration service to track all currently hosted agents, (3) management services that monitor external control of agents, (4) transport services for migration of agents from one agency to another, (5) security services for protecting remote interactions and agency resources, and (6) persistence services for enabling the storage of agents for possible recovery. *Agents* are computer programs characterized by a set of attributes. They can be mobile or stationary. Mobile agents move from one location to another within a region to take advantage of local interactions, and thus are capable of reducing network load by migrating. Stationary agents are associated with a particular location only, and are incapable of migration.

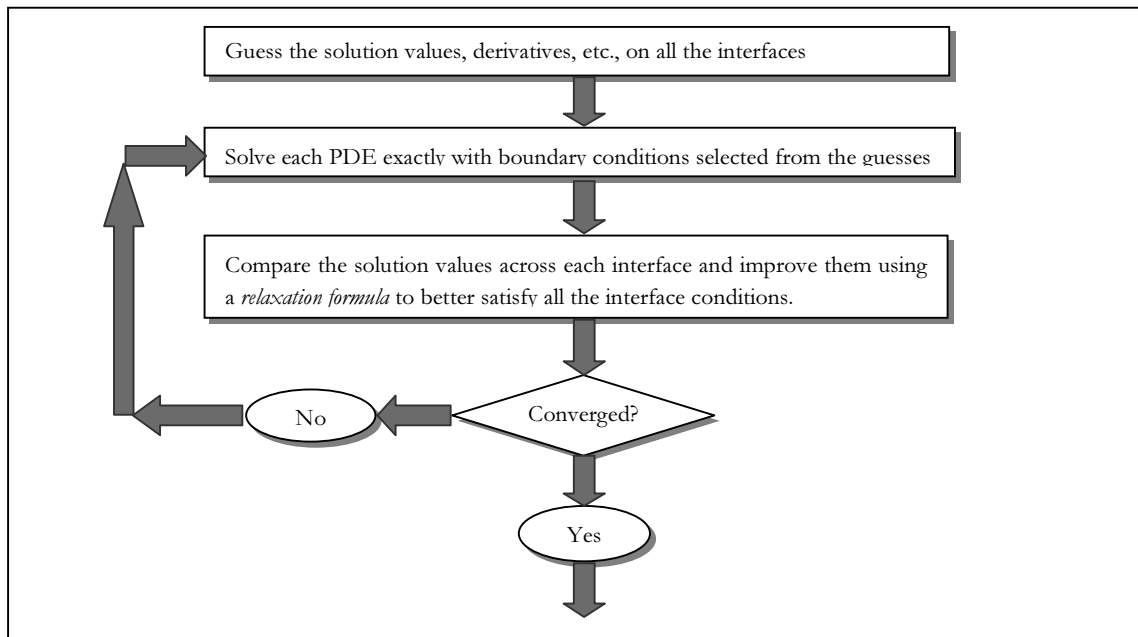


Figure 3: The interface relaxation iteration

THE MPSE AGENT-BASED DESIGN

The Grasshopper environment supports the traditional client-server structure. MPSE clients and servers are Grasshopper agents which provide modeling (specification of the physical object to be simulated), simulation control and computational service. The MPSE framework consists of 6 classes of agents (Figure 4): the ModelingAgent, the SimulationControlAgent (SCA), the VisualizerAgent (VA), the LegacyCodeAgents (LCA), the MediatorAgents (MA), and the ServiceAgents. The ModelingAgent and SCA are client agents; the remaining classes of agents are servers. The ModelingAgent, SCA and VA have graphical user interfaces which support user interaction. The ServiceAgents provide services exclusively for the SCA, such as resource identification and database access for agent information retrieval.

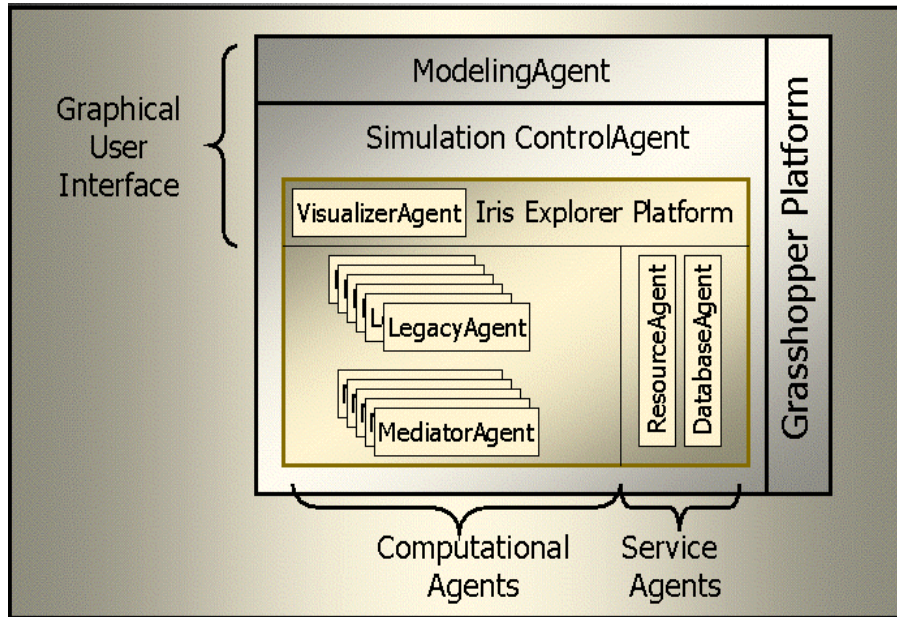


Figure 4: The agent-based design of the MPSE. The *SimulationControlAgent* is the client which controls the entire simulation process. Legacy and mediator codes (generally Fortran or C programs) are transformed into server agents. They communicate boundary interface information using the SCA as an intermediary. Users interact primarily with the Modeling and Visualizer Agents. All agents are built and executed within the Grasshopper environment.

The *ModelingAgent* is the top level user interface. It allows users to define the target simulation object (domain) and its geometric decomposition into subdomains. Users can then select the configuration of subdomains to be simulated. These subdomains are passed to grid or meshing tools (such as TrueGrid [71]), which are incorporated into the *ModelingAgent* as self-contained systems. The meshing tools are used to generate the grid and define the boundary conditions for the user-selected subdomains. The *ModelingAgent* assists users to identify the interfaces between the domains, to define data exchange information for each interface, and to select solvers for each subdomain. This information becomes the startup data for the SCA.

The SCA requests and manages the services of all server agents. It controls the entire simulation process: launching/terminating the computational servers, managing their interactions, and facilitating the cross-network asynchronous communication of computational and control data. The VA is a server agent that receives solution data and renders it graphically using the IRIS Explorer data visualization system [70]. The LCAs are the computational agents that simulate the subdomains of the decomposed physical object; each LCA encapsulates an established legacy code targeted for a specific physical sub-object. The MPSE iteration algorithm requires an LCA to communicate its boundary data (via the SCA) to one or more MAs. The MAs encapsulate the mediation codes that are responsible for adjusting and resolving the interfaces (represented by the boundary data) between neighboring subdomains, each of which is simulated by one LCA. The SCA can handle any number of subdomains; i.e., it can control any number and type of communicating legacy code and mediator agents.

In the following sections, we describe a template procedure for creating agents from legacy and mediator codes, and then discuss the generic operation of the SCA.

IMPLEMENTATION OF A LEGACY CODE AGENT

The initial challenge was to create a template procedure for embedding legacy Fortran code into the server agent structure. The resulting LCA could then exercise control over the Fortran code and enable the data flow by (1) starting up the legacy code, (2) pausing after each Fortran iteration to communicate the required boundary data to the SCA, (3) receiving the mediated boundary data in return, (4) continuing with the next iteration, and (5) signaling to the Fortran code that the process should terminate so that the final solution output files can be generated.

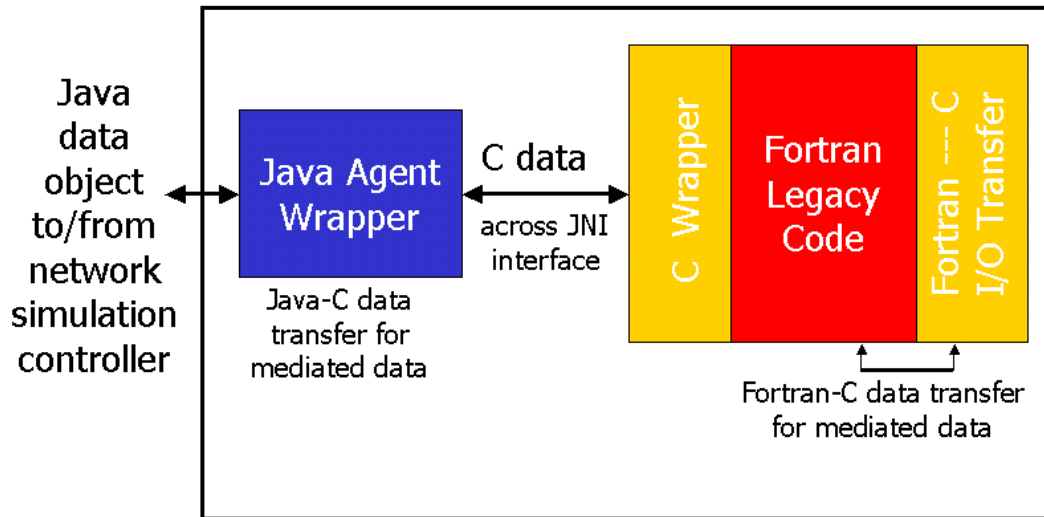


Figure 5: Generic template procedure for embedding legacy Fortran code into a Java Grasshopper agent. This procedure transforms a Fortran executable into an LCA legacy library, which can be invoked and controlled by the LCA.

The following template procedure was developed to transform legacy code, which generally starts out as a Fortran executable, into a C-wrapped legacy code library as follows (Figure 5):

- Change the Fortran “main” to a subroutine. Command line arguments are passed as parameters from the C-wrapper,
- Start the Fortran subroutine as a thread from the C-wrapper routine,
- Define C structures to hold the data representing boundary information,
- Write a C data transfer routine to copy boundary data back and forth between the C and Fortran data structures,
- Insert a call to the C data transfer routine in the Fortran iteration loop, and
- Insert control code in the C-wrapper to sleep/wake the C data transfer routine, thus effectively controlling the pause and restart of the Fortran iterations.

The C code for copying the data (both in the wrapper for the conversion from Java object to C, and in the data transfer routine for conversion from C to Fortran) requires a number of source code lines corresponding to the amount of data to copy. Less than 20 lines of C code are required for starting the thread and controlling the start/stop of the Fortran iterations. The changes to the Fortran code are minimal. The C-wrapper is defined with a JNI (Java Native Interface) interface to the Java agent

code, and the C boundary data is passed up through the JNI interface as function parameters to the Java agent. The modified Fortran source code and the two new C routines (the wrapper and the data transfer routine) are compiled together as a *legacy library*, which is loaded into the Java LCA server when it is instantiated. The Java legacy agent starts the C-wrapper and waits for the JNI object containing the boundary data. When the data object is received, the agent serializes it and communicates it to the SCA. The SCA passes the object to the mediator, and returns the mediated data object to the agent, which copies it into a JNI object and passes it down to the C-wrapper. The SCA is also responsible for passing the LCA the number of iterations when it is instantiated, and the LCA passes this information to the legacy code so that the iteration process can terminate properly.

IMPLEMENTATION OF A MEDIATOR AGENT

Mediators are customized Java or C programs that are easily encapsulated as Java agents. The mediation code is wrapped by agent code that handles the boundary data communication between the MediatorAgent (MA) and the SCA. In general, the mediator code is heavily dependent on the legacy codes and the data types that are exchanged between the two legacy codes. Each shared boundary (interface) between two subdomains requires an MA to mediate the boundary data. This mediation code must take into account the computational requirements of both solvers as well as the geometry (grid) of the domains on both sides of the boundary. The mediator must interpolate the grid points for the data exchange, since the grid points on the interface boundary for the two domains are not the same.

THE SCA AND SIMULATION CONTROL

The SCA is responsible for managing the entire simulation in accordance with the specifications provided by the user through the ModelingAgent. Users select the subdomain (physical parts) to be simulated, generate geometry and parameter files for these parts, identify the length of the simulation process (in time units), and specify the output requirements. The SCA determines the software and hardware components to be used in simulating the specified configuration, that is, the SCA selects the LCAs and MAs to be instantiated and chooses the machine resources on which they will run. The SCA also translates the simulation time specification into iteration specifications for each LCA, computing an iteration correspondence factor to ensure that the simulation time when boundary information is exchanged will match for solvers on both sides of the interface.

Since the SCA is the intermediary for data exchange, it must contain code that is customized for the data structures involved for a particular MPSE. The data exchange routines, however, are the only part of the SCA which is MPSE specific. The remaining code is the standard asynchronous client agent sample code provided with the Grasshopper system. The SCA client agent invokes the LCAs and MAs, receives data from the LCAs, coordinates exchanges for each interface (between two LCAs and their corresponding MA), returns mediated data to the LCAs, and sends data to the VisualizerAgent when requested to do so. The entire simulation process begins and ends with the SCA.

SECTION 5: GASTURBNLAB: AN AGENT-BASED MPSE

We now present the implementation of GasTurbnLab using the framework described in the previous section. We identify two existing legacy code PDE solvers to be encapsulated as LegacyCodeAgents, and we write the customized interface mediation code for two MediatorAgents to support the exchange of interface boundary data between the LCAs.

THE GASTURNLAB LEGACY CODE AGENTS

Two legacy codes have been encapsulated as LCAs: *ALE3D*, an advanced CFD code for simulating turbines, and *KIVA*, an advanced combustion simulation code. The *ALE3D* solver models the stator and rotor parts of the engine compressor, and the *KIVA* solver modules the engine combustor. The two agents created by encapsulating these solvers, the *AleAgent* and the *KivaAgent*, have been used successfully to simulate several engine configurations for stator rows, rotor rows and combustors. The solutions generated by the simulations have been validated, and the results are described in section 6. Simulations for more complicated configurations that answer questions about stall, surge and fatigue are also described; these simulations are currently underway.

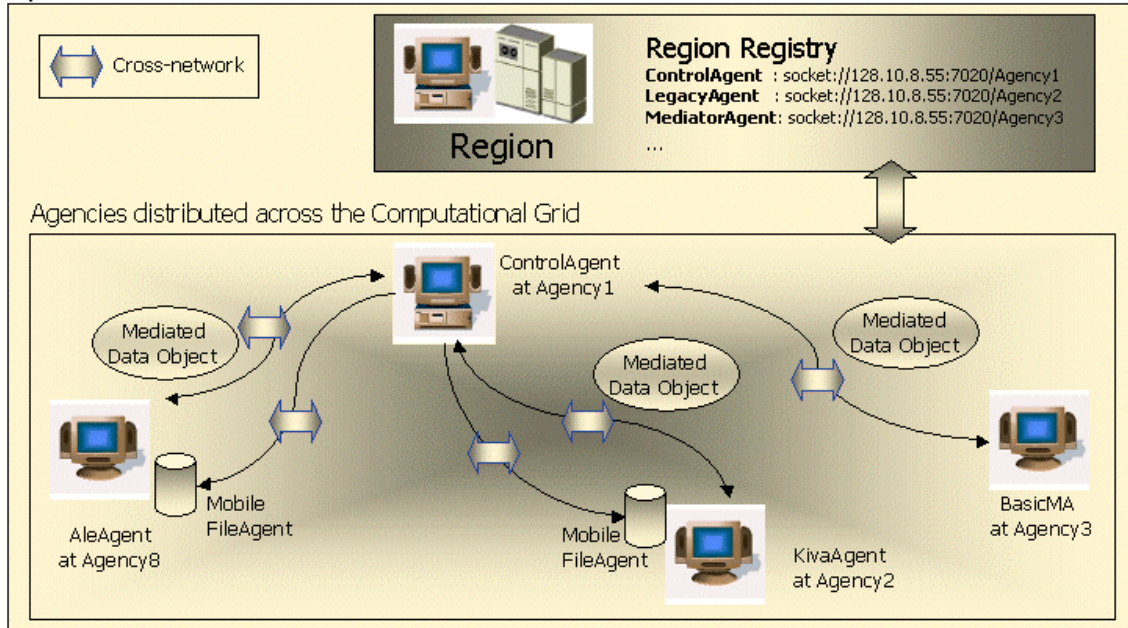


Figure 6: Sample simulation scenario from the GasTurbnLab MPSE. The scenario shows a stator-combustor two domain simulation, mediated at the outflow boundary of the stator and the inflow boundary of the combustor by the BasicMA mediator agent.

THE GASTURBNLAB MEDIATOR AGENTS

Mediation code is MPSE specific code that handles the exchange of boundary data on the interface between two domains. Two MAs, a *basic* mediator and an *advanced* mediator, were developed for mediating the boundary data for the GasTurbnLab domain interfaces. Both MAs are capable of mediating inflow or outflow interfaces for any combination of *ALE3D* and *KIVA* engine parts (domains). Either mediator can be used to obtain accurate solution results for problems with no unsteadiness (i.e., steady state problems). The difference between the mediator codes is in the selection of boundary data used to mediate the interfaces; for unsteady problems, the advanced mediator (*AdvMA*) provides a more accurate solution than the basic mediator (*BasicMA*.) The *BasicMA* was created as a simple solution to the boundary interface problem. Based on this experience, the *AdvMA* was created to provide a more accurate transfer of unsteady fluid mechanic phenomena across the interface.

The *BasicMA* treats the boundary between domains in the same way inflow/outflow boundaries are treated when open to the atmosphere, i.e., when an inflow or outflow boundary is open to the atmosphere, atmospheric conditions are specified at the beginning of the simulation. The *BasicMA*

simply updates these conditions after each time step based on the aerodynamic data from the adjacent domain. Except for the data being passed to them from the adjacent domain, the inflow/outflow boundary algorithms are the same whether the boundary is connected to the open atmosphere or to another ALE3D or KIVA domain. The drawback of this technique is that the inflow/outflow boundary condition algorithms are inherently less accurate than the interior algorithms. This mediator, however, has been shown to produce acceptable results on ALE3D-KIVA interfaces.

The AdvMA calculates elemental and nodal quantities at the interface between domains in nearly the same way as the interior elemental and nodal quantities are calculated. When the AdvMA is used, the inflow/outflow algorithms are bypassed and more accurate algorithms are employed. These algorithms are nearly identical to the interior algorithms. Specifically, there are two areas in which the AdvMA provides more accurate values at the interface between the two domains. First, ALE3D and KIVA codes both use a second-order upwinding scheme for the calculation of density and energy fluxes of interior faces during their advection steps. The AdvMA maintains second-order upwinding at the interface between the legacy codes while the BasicMA does not. Second, the AdvMA more accurately handles the exit boundary of a domain during the Lagrange calculation when it is connected to another domain. Unlike the BasicMA, the AdvMA provides forces at the exit so that the acceleration of exit nodes are computed the same way as interior node accelerations.

THE GASTURBNLAB SCA

When a user invokes the GasTurbnLab SCA from the Grasshopper graphical interface, the modeling specifications from the ModelingAgent are passed as input. The specifications describe the configuration of combustors, stator rows and rotor rows, listing the interfaces between these engine parts which are to be mediated. In addition, the geometry and solver parameters for each engine part, as well as the total simulation time are identified. Each engine part is simulated by one LCA, and each interface between two engine parts is mediated by one MA. After verifying that the required number of LCA and MA servers are available, the SCA determines which servers (i.e., which agents on which machines) are to be instantiated.

The SCA creates a KivaAgent proxy for each combustor, an AleAgent proxy for each stator and an AleAgent proxy for each rotor. The SCA then determines which MAs are required to mediate the interface boundaries identified by the user. The BasicMA mediates any interface involving a KivaAgent; the AdvMA mediates interfaces where two AleAgents exchange boundary data. The required number of AdvMAs and BasicMAs proxies are created, one per identified interface. The geometry and parameter files for the solver startups are passed to the LCAs via mobile FileAgents. The SCA uses the specified simulation *time in ms* and the *time step* information from all solver parameter files to compute the number of iterations required for each LCA. Since the KivaAgent time steps are generally much larger than the AleAgent time steps, the SCA determines a coordinating factor that allows the ALE3D iterations and the KIVA iterations to exchange data when the simulation time values match. For a given ALE3D-KIVA interface, the AleAgent may direct the ALE3D legacy code to iterate 10, 20 or even 60 times before sending interface data, while the KivaAgent directs the KIVA code to iterate once. The simulation time at the exchange is then equal for each code. The SCA is responsible for computing and communicating these factors to all LCAs so that the exchanges occur properly as the simulation time advances.

The SCA continues to facilitate the data exchange during the entire simulation process, receiving data from the LCAs and MAs, and sending them to their proper destinations. When the final simulation time step is finished, each LCA terminates its legacy code (ensuring that the proper exit processing

occurs) and sends a termination signal to the SCA. The SCA shuts down the entire simulation when all LCAs have terminated. It then allows the user to access the VA for data visualization.

SECTION 6: GASTURBNLAB: SIMULATION RESULTS

Five configurations of engine parts consisting of stators, rotors and combustors have been simulated. Each explores the result of connecting the inflow or outflow domain boundary of one engine part to the boundary of another engine part for data exchange via mediation. The first two configurations were simple two domain simulations. In the remaining three configurations, the number of engine parts and the complexity of the boundary connections were incrementally increased.

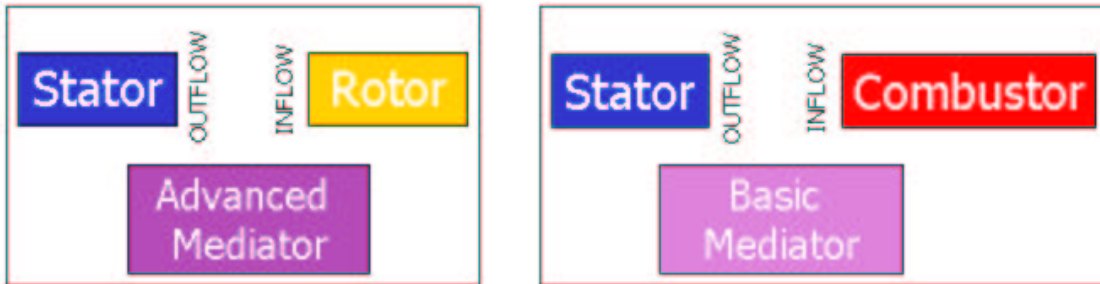


Figure 7: Block diagram of the stator-rotor simulation using two AleAgents (left) and the stator-combustor simulation using one AleAgent and one KivaAgent (right).

The first configuration connected an upstream stator (AleAgent) to a downstream combustor (KivaAgent), using the BasicMA to mediate the data along the interface for stator outflow and combustor inflow. This simulation was run for several variations of geometry, initial pressure conditions and number of iterations. The agent diagram for this simulation is shown in Figure 6. A block diagram showing only the engine parts and their boundary connections is shown on the right in Figure 7. Results of the stator-combustor simulations are shown in Figures 8-10.

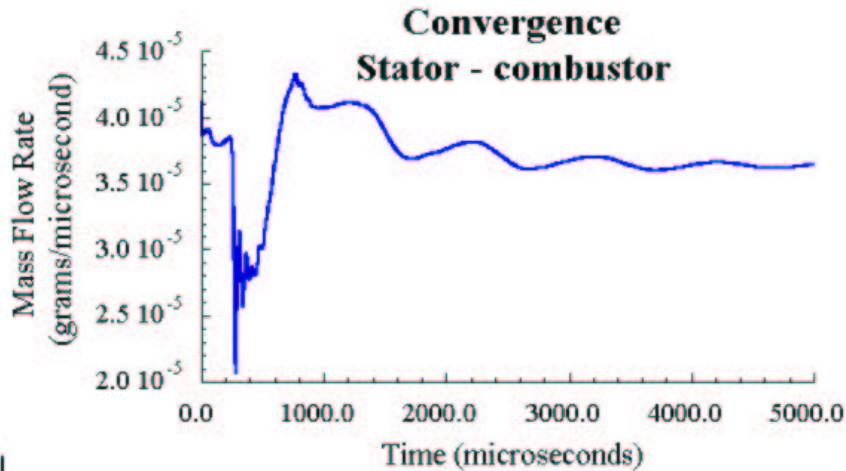


Figure 8: The mass flow rate moves towards a steady state value after 500 KIVA time steps (5000 corresponding ALE3D time steps) in the stator-combustor simulation.

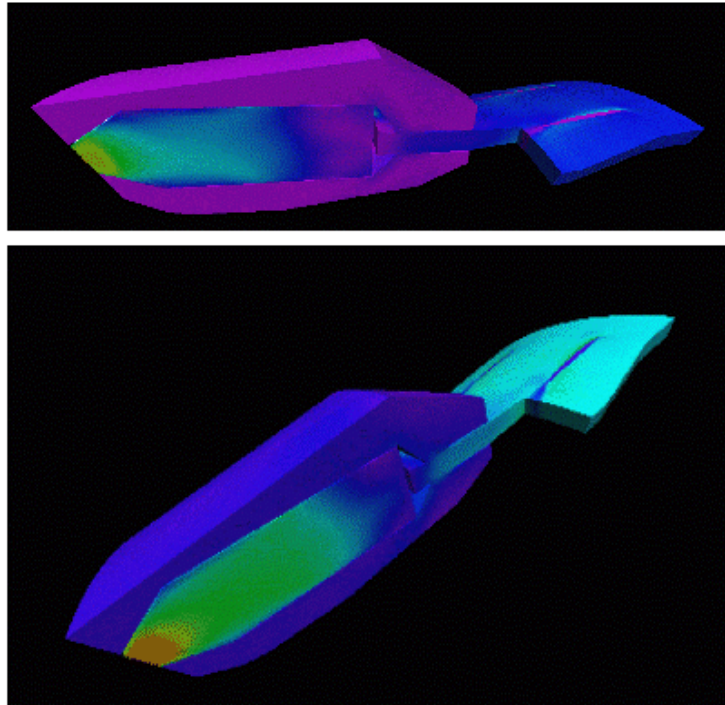


Figure 9: Plot for speed (top) and z velocity (bottom), with the stator upstream and the combustor downstream. The simulation involves an *AleAgent*, a *KivaAgent* and the *BasicMA* basic mediator.

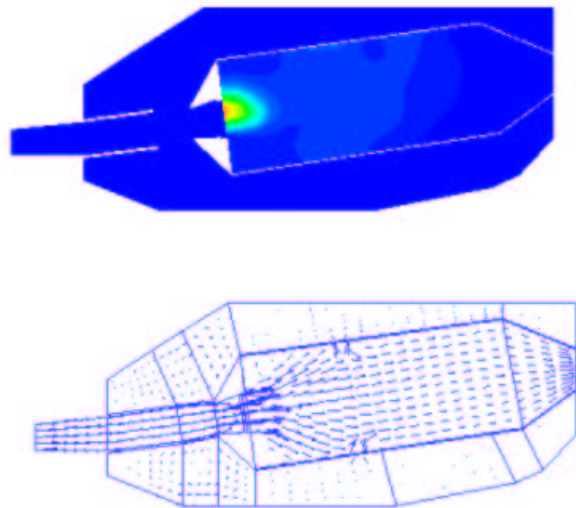


Figure 10: Combustor plots which show the temperature contour indicating an ignition process in the beginning of combustion (top) and the velocity vector field (bottom). The results are from a stator-combustor simulation.

The second configuration connected an upstream stator to a downstream rotor and was simulated using two *AleAgents*. The interface boundary was mediated by the *AdvMA*, and involved the exchange of force, flux and velocity data, with three mediator exchanges per time step. This simulation ran for more than 16,000 iterations. The final mass flow was $6.25e-5$ grams/microsecond at stator inflow and $6.32e-5$ grams/microsecond for rotor outflow. The total pressure ratio was 1.480. A block diagram of the configuration of these engine parts and their interface boundary is shown on the left in Figure 7; plots of the results are shown in Figures 11 and 12. Figure 12 shows the *GasTurbnLab VisualizerAgent* with the integrated *IRIS Explorer* environment.

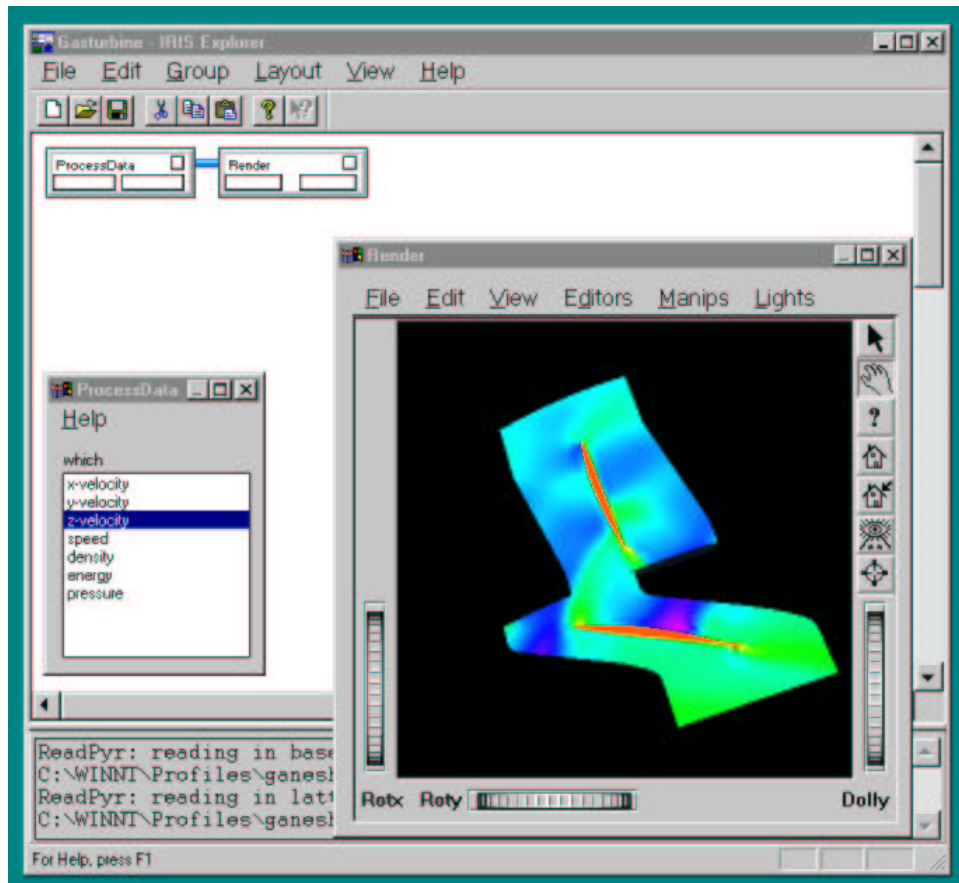


Figure 11: User interaction with the *VisualizerAgent*. The plot shows the results of a stator-rotor simulation for two *AleAgents* and one *AdvMA* mediator agent.

The block diagrams in Figure 13 show three engine configurations consisting of more engine parts with additional complexity on the interface boundaries. Some engine parts in these configurations involve two mediations, one for the inflow boundary and one for the outflow boundary. This additional complexity is completely scalable for the *SCA*, and adds no complexity to its operation. Each *LegacyCodeAgent* knows when its current simulation time requires it to exchange data, and it handles the preparation of the data object and its communication to the *SCA*. The *SCA* functions only as router, knowing the source and destination for each data object it receives.

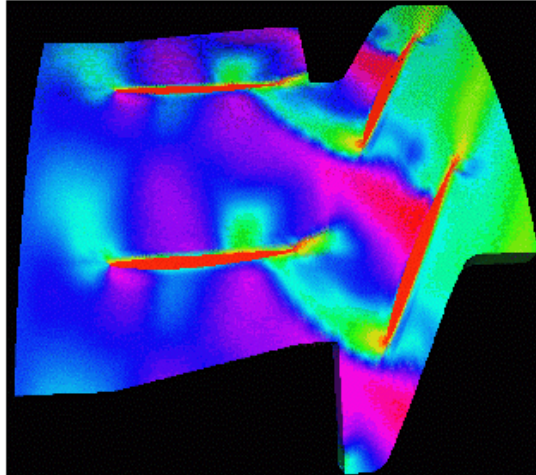


Figure 12: Plot of a row of stators upstream and a row of rotors downstream after 10000 iterations. The mediation along the interface boundary involves force, flux and velocity data, with three mediator exchanges per time step.

The simulation of the configuration in the diagram at the bottom of Figure 13 (shown in simplified form without inflow/outflow markings and mediator blocks) is currently underway. Results from this simulation are expected to answer questions about engine stall.

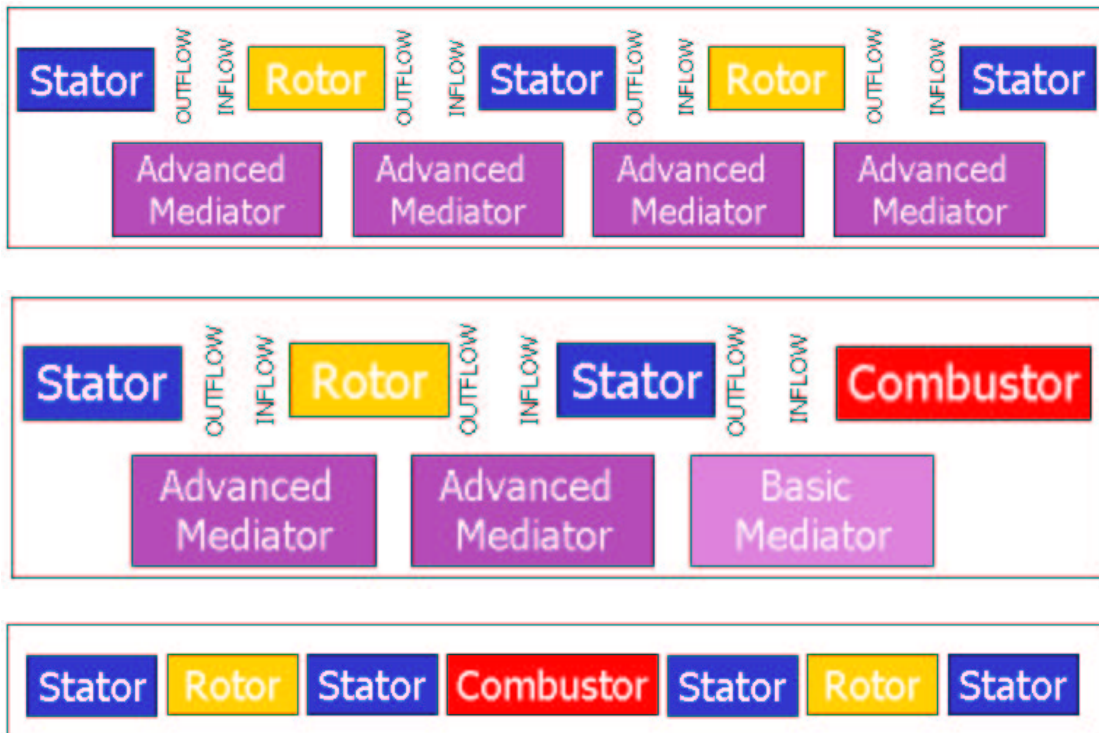


Figure 13: Three block diagrams of engine configurations, with added complexity in boundary connections and number of engine parts.

The major goal of the simulations discussed in this section was to verify the results of the various configurations of engine parts, thus validating the multi-physics interface relaxation method and the

agent-based design. In addition, some informal timing data was gathered for these five simulation cases. In general, the system operates at about 150 KIVA iterations per hour (corresponding to about 1500 ALE3D iterations) when each agent is placed on its own machine. It is interesting that the KIVA code running in its original form as a stand-alone Fortran executable also executes at about 150 iterations per hour. Thus, it appears that the overhead of our MPSE agent-based infrastructure does not degrade the performance of the PDE solvers.

SECTION 7: GASTURBNLAB: PROPOSED SIMULATION SCENARIO

To study complex physical phenomena such as stall, surge and turbine blade fatigue, the GasTurbnLab configurations must consist of more engine parts and more complicated boundary connections than those described in the previous section. Current simulation configurations have involved no more than seven engine parts, and all boundaries were connected at the inflow/outflow interface with the two neighboring domain geometries corresponding exactly at the interface boundary.

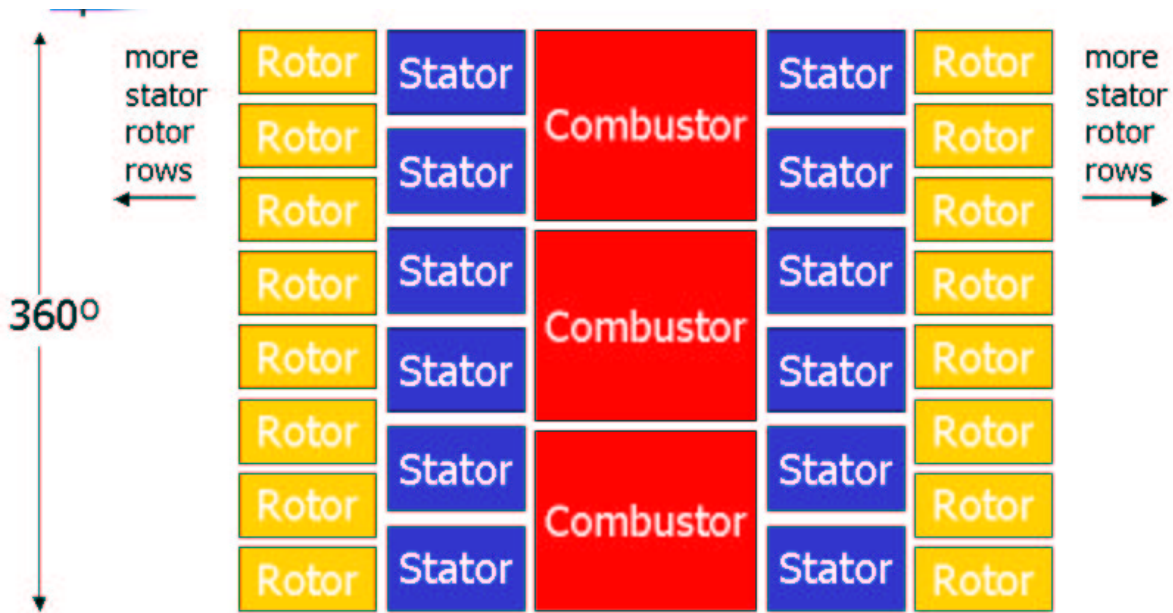


Figure 14: Configuration of the final simulation for the gas turbine engine compressor - combustor.

In Figure 14, the configuration consists of dozens of engine parts. The boundary geometry of a single engine part may overlap two or more interface boundaries in the adjacent domains. The restriction of inflow/outflow interface connections has been removed, and domain interfaces now involve side-to-side boundaries where periodic boundary conditions were previously applied. None of the additional complexity affects the design of the framework, the operation of the SCA or the functionality of the agents, since the design of the framework and the MPSE agents are scalable to any number of domains. Most of the changes will be in the mediator code, which must now handle the added complexity of the interface geometries. The legacy code itself will need to change in order to replace code involving the periodic boundary conditions with iterated data from adjacent domains. The changes will be made incrementally, as demonstrated in the incremental configuration changes shown in Section 6, since the physical results for each new configuration must be carefully validated before moving to the next level of complexity.

SECTION 8: CONCLUSION

In summary, we have described an agent-based framework for building multidisciplinary problem solving environments for complex, large-scale simulations. The design is based on the geometric modularity of the simulation computations.

The MPSE framework uses the Grasshopper Agent Platform for its agent-based infrastructure, making heavy use of the flexible and robust agent creation and distributed execution features of the Grasshopper environment. The MPSE framework can be extended at the user interface level by adding additional modules that support user interaction for model specification or data visualization. Furthermore, the framework can be extended at the enabling services and computational levels by creating new agents which perform additional services or computations. To facilitate legacy code incorporation, we have proposed a simple and very general wrapper method for encapsulating Fortran or C code as an agent library.

The GasTurbnLab MPSE is an implementation 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 technique outlined in Section 4. A suitable load balancing algorithm can be implemented within the SCA for better distributed performance of the highly compute intensive simulations. The ModelingAgent 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 GasTurbnLab with suitable pre- and post-processing modules for each target gas turbine simulation problem.

The proposed MPSE framework architecture is scalable, enabling the implementation of very large scale, distributed problem solving environments for scientific simulations. It is also versatile and simple enough to be used to build prototype problem solving environments to analyze and validate mathematical techniques for interface relaxation. Thus, it is a useful environment that advances 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. Kluever 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. Engrg.*, **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 Intelligence*, (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>.