International Journal of Computational Engineering Science
Vol. 1, No. 1 (2000) 33-60

AN AGENT-BASED NETCENTRIC FRAMEWORK FOR MULTIDISCIPLINARY PROBLEM SOLVING ENVIRONMENTS (MPSE)

S. MARKUS, E. N. HOUSTIS, A. C. CATLIN, J. R. RICE, P. TSOMPANOPOULOU, E. VAVALIS*, D. GOTTFRIED, K. SU, and G. BALAKRISHNAN

Dept. of Computer Sciences, Purdue University,
1398 Computer Science Build., W. Lafayette, IN 47907-1398, USA
E-mail: {markus,enh,acc,jrr,giwta,mav}@cs.purdue.edu,
gottfrie@ecn.purdue.edu, KeSu@calumet.purdue.edu, balakrig@purdue.edu

The process of *prototyping* is part of every scientific inquiry, product design, and learning activity. The new economic realities require the rapid prototyping of manufactured artifacts and rapid solutions to problems with numerous interrelated elements. This, in turn, requires the fast, accurate simulation of physical processes and design optimization using knowledge and computational models from *multiple disciplines (multi-physics and multi-scale models)* in science and engineering. Thus, the realization of **rapid multidisciplinary prototyping** is the new *grand challenge.* In this application scenario the natural computational resource is a "computational grid" that connects the needed distributed hardware and software resources used to simulate the elements of the artifact. Our research goal is to address this application scenario in the context of parallel computing, cluster computing (LAN based computational grids), and Intranet/Internet computational grids. In this document, we describe the initial design of a generic MPSE framework based on a network of computational agents assuming 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.

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)"[5,6] promises to contribute toward the realization of this prediction for physical modeling and to provide students, scientists,

*Author's permanent address: University of Crete, Dept. of Mathematics, 71409 Heraklion, Crete, Greece.

and engineers with environments that allow them to spend more time doing science and enginerring 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 (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 to as GasTurbnLab, for the simulation of gas turbine engines. In Sec. 5, we describe the application software infrastructure in the GasTurbnLab prototype. In Sec. 6, we describe the architectural components for a generic MPSE framework, along with issues pertaining to the GasTurbnLab instantiation of this MPSE framework. In Sec. 7, a prototype implementation of the GasTurbnLab MPSE is described. We conclude our discussion in Sec. 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 the GasTurbnLab prototype.

2. MPSES – Definitions and Research Issues

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

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 21^{st} century: 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[1,9]". 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[24]" (PDEs). 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[26]", ii) "a knowledge based framework to address computational intelligence issues for PSEs[10,16]" and for PDELab, iii) "infrastructure for solving PDEs[11,12,13,23,25]", and iv) "parallel PDE methodologies[2,17,18,27,28,29]" and "virtual computational environments[4,15,31]".

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.

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[19,20,21,22]" have been addressed 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[3]" 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 a 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 desired), 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 a prticular design 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.

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 Fig. 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[6]". 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. The hardware infrastructure assumed for these simulations and the implementation of MPSE consists of a computational grid involving a SP-2, 128 PC cluster running Solaris, and SGI Origin 2000 with 32 CPUs. In this study we will utilize the agent system "*Grasshopper*[32]" that is MASIF (Mobile Agent System Interoperability Facilities Specification) standard compliant and runs on the top of CORBA. Details of this implementation follow.

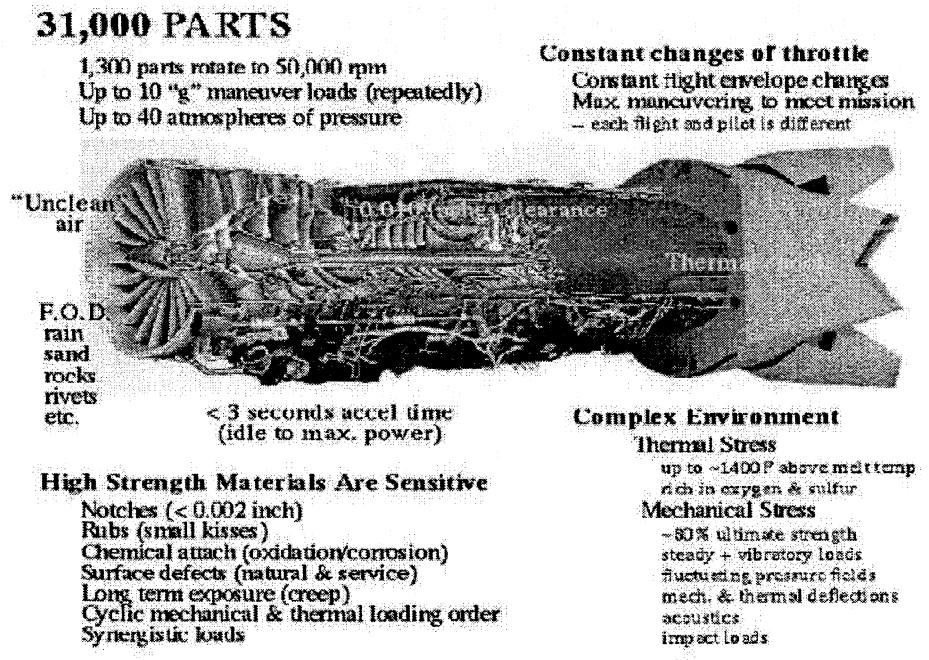

Fig. 1. View of a gas turbine showing some of its detail, some of its operational characteristics and the engineering methodologies involved in its design, simulation and construction.

4. GASTURBNLAB: A Prototype MPSE Framework For Gas Turbine Engine Simulations

In this section we describe the "design of a MPSE framework[5]" that can be used to simulate complex multi-physics phenomena governed by PDE network models in general and the requirements of the GasTurbnLab MPSE. 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 is then solved independently, with mediator interactions along the boundaries for "interface relaxation[19,20,22]". 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. Figure 3 gives a brief overview of this simulation paradigm. 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.

4.1. *Functional specifications of GASTURBNLAB*

In the case of PDE simulations, the MPSE framework user interface is driven

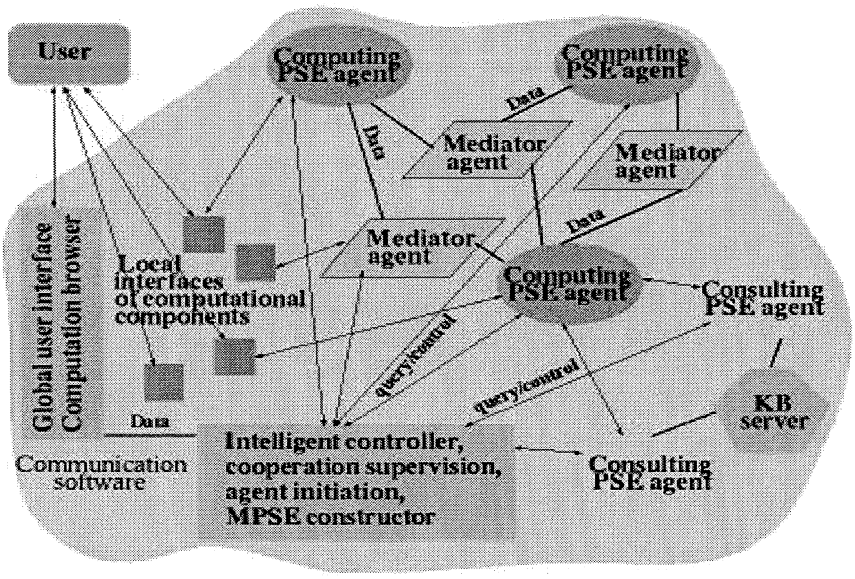

Fig. 2. Functional view of a multidisciplinary PSE. The computations (and the major data exchange) are concentrated in the network of solver (PSE) and mediator agents. The solver agents communicate with the recommender ones through queries to obtain "advice" on computational parameters. The user interacts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

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 overheard 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.

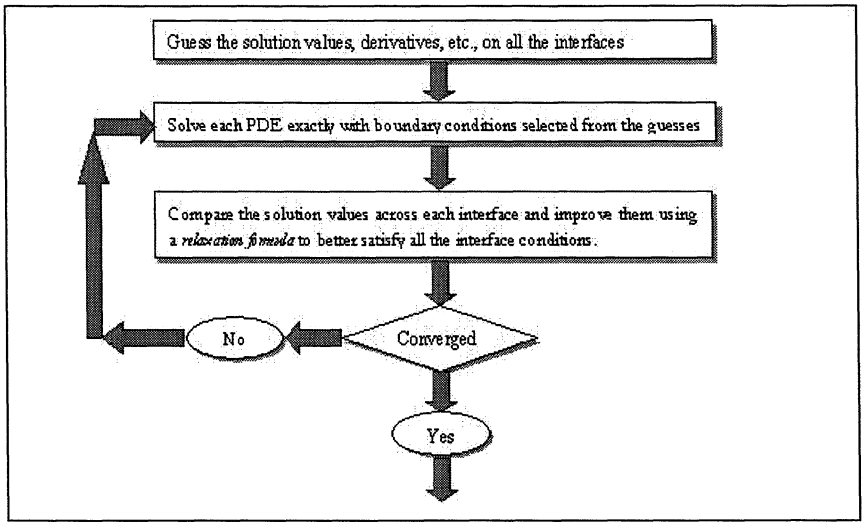

Fig. 3. Interface relaxation iteration.

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. In the GasTurbnLab MPSE, the initial target object is the entire gas turbine engine. A GasTurbnLab simulation 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.

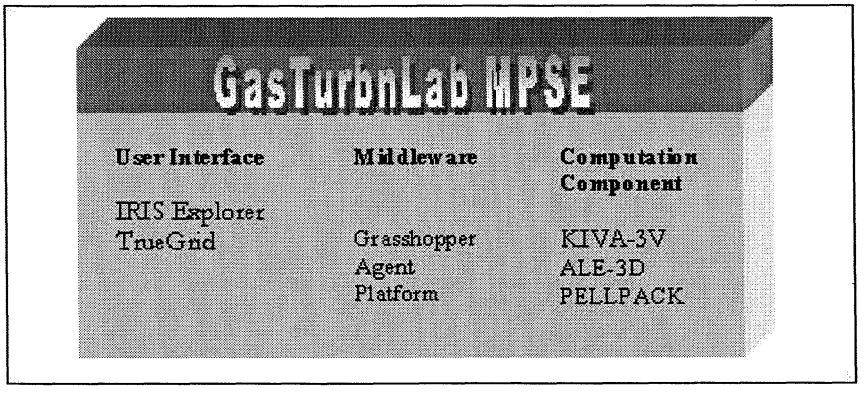

Fig. 4. Major components of the GasTurbnLab MPSE.

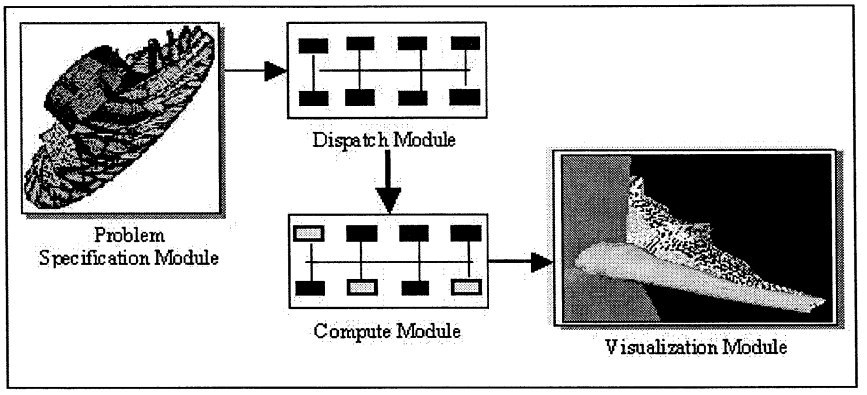

Fig. 5. IRIS Explorer top level user interface for the GasTurbnLab MPSE.

4.2. *Enabling technologies and 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 atchitectural components - *the user interface layer, the middleware, and the computational software infrastructure* layer. The "IRIS Explorer application builder and visualization system[33]" is used for the MPSE framework user interface and the middleware component is based on the "Grasshopper[32]" mobile agent platform. The computational infrastructure is dependent upon the MPE's target class of simulation problems. This computational application software infrastructure is discussed in Sec. 5. Figure 4 depicts these architectural layers and their major constituents for the GasTurbnLab PSE.

4.3. *Graphical user interface*

The IRIS Explorer system is a toolkit for building user interfaces for data visualization, and uses a data flow paradigm. The interface is built by creating the requisite modules and wiring them together via Explorer's map editor. The connections in an Explorer map depict the flow of data between modules and act as module triggers (Fig. 5). 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. The SKM language with Lisp-like syntax is used to create these scripts for the Explorer command interface.

The Explorer interface provides access to all MPSE framework components that are user-steerable, including problem specification, simulation launch and control, and solution visualization. Users define the target simulation object (domain) and its geometric domain decomposition with the *Problem Specification Module.* User-selected subdomains of the target object are passed to the "TrueGrid[34]" software tool, which is incorporated into the module as a self-contained system. TrueGrid is applied to each subdomain to generate a grid and define boundary conditions. The Problem Specification Module provides tools for specifying interface conditions between the subdomains and assigning solvers and their parameters to each subdomain. This information is the required input for the *Dispatch Module.*

Users launch the simulation via the *Dispatch* and *Compute Modules*, which interact with the underlying Grasshopper agent platform. Grasshopper's graphical monitoring tool is used within the Compute module to view and monitor the underlying agent interactions in the simulation process for possible computational steering. Once the Compute Module completes its simulation task, control is returned to the Explorer interface for the solution visualization and analysis phases. Explorer's data visualization module, Render, is utilized by the Visualization Module for post-simulation solution display and analysis. Render is a built-in Explorer module which displays a geometric object, rendered from a 3D geometry structure sent to it by a compatible preprocessing module. Render provides a wide selection of visuak enhancement techniques for manipulating and examining a display object. The GasTurbnLab *Visualization Module (Visualizer)* is the first Explorer module to be implemented. A brief description of its implementation and some solution images from the prototype simulations (presented in Sec. 6) follow. The Visualizer displays the final solutions for the simulation of any number of subdomains by any of the integrated legacy solvers. The *Visualization Module* can be viewed (Fig. 6) as an integrated system of preprocessing programs, preprocessing modules and Render. The Visualizer's preprocessor recognizes a standardized file format for node and element data which has been defined for the solvers of the GasTurbnLab MPSE; all solvers are expected to write their final results to a solution file in this format. Solution files contain vertex and element connectivity information, along with nodal and element based solution values. The files are presented to the Visualizer, which invokes the preprocessor for

- Loading the solver-generated data,
- Building a pyramid data structure as required by Explorer,
- Selecting which solution values to display, and
- Passing the data structure and display parameters to Render.

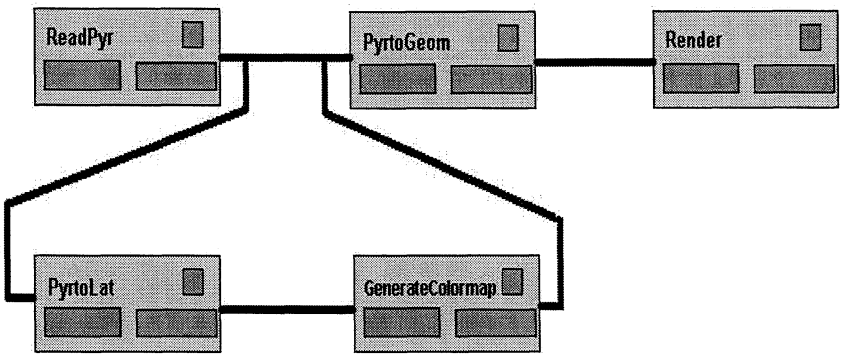

Fig. 6. A portion of the Visualizer implementation, shown as a wired map of modules in IRIS Explorer. Each module has an input port, and output port and a parameter dialog.

When data is loaded from the solver generated files, the preprocessor must normalize values from different solvers so they can be presented as a single image. Additional image processing is necessary to handle large differences in solution values between solvers, data values with an extremely small range, solution representation differences, etc. The pyramid data structure created for the GasTurbnLab simulation output data is a layered structure which contains the information required for the image display: vertex (nodal) coordinates, vertex solutions, edges (connections between nodes), edge based solutions, surfaces, surface based solutions, solids (3D elements), element based solutions. When the pyramid structure is complete, internal Explorer modules are used to transform it to the Render geometry data structure. Finally, the Visualizer creates a user selection panel allowing users to choose which solution to display. The prototype simulation discussed in Sec. 6 generates nodal based velocities (in the x, y and z directions) and element based density, energy and pressure. Users choose which of these solutions to pass to Render. Figure 7 shows some Visualizer images from the prototype simulation.

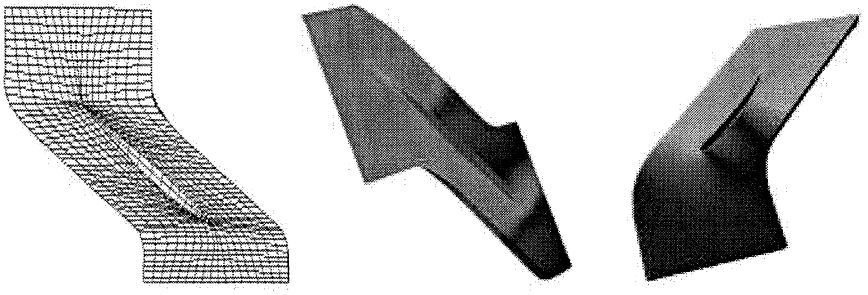

Fig. 7. Images from the GasTrubnlab MPSE for the rotor simulation. To the left is the TrueGrid domain specification. The middle and right images are Visualizer images, displaying the mediated ALE-3D solution.

4.4. *Middleware*

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

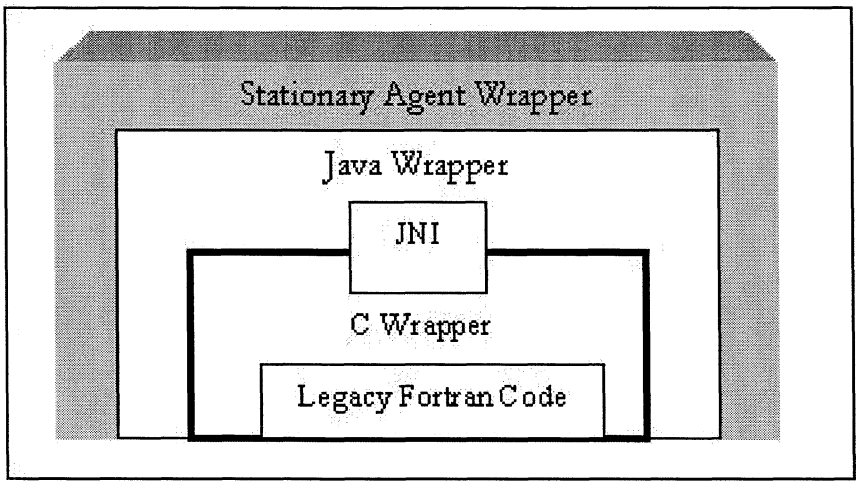

Fig. 8. Anatomy of a legacy code embedded stationary agent.

4.5. *Application software*

The computational infrastructure in the MPSE determines its target class of problems. The proposed framework provides the architecture and model infrastructure for an agent-based simulation MPSE and facilitates a straightforward incorporation of computational code to GasTurbnLab. The 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 consists of ALE-3D, KIVA-3V, and PELLPACK code modules and interface relaxation code implemented in either C/C++ or Java. ALE-3D is an advanced CFD software module suitable for gas turbine simulation. It is large, with about 200,000 lines of code. KIVA-3V is an advanced combustion-simulation package with about 50,000 lines of code. PELLPACK is a versatile PSE

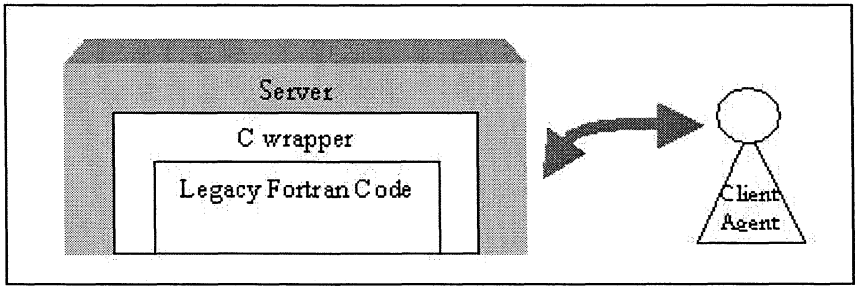

Fig. 9. Client /server approach for legacy code encapsulation.

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.

(i) Insert the Fortran-based code within Java wrappers as stationary agents. This can be achieved with JNI (Java Native Interface). Figure 8 illustrates the encapsulation technique within a stationary agent.
(ii) Insert the Fortran-based code within C/C++ wrappers as servers. They can then be accessed as local servers by client agents. Figure 9 illustrates the legacy code embedded server and the client agent interaction.

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 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 allows legacy code incorporation based on either of these two approaches.

5. 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 (Fig. 10) 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 are 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.

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 controls the local database on the host. It has 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 are stored in an XML format based

on a proprietary DTD (Document Type Definition). Such a specification is only 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 is 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. The RA agent monitors execution performance and gather local machine load and network congestion information. It maintains a local resource database along with other requisite logs. The local RA synchronizes its resource information with resource agents on other hosts. Thus, each RA has 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. This event model for resource monitoring facilitates 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.

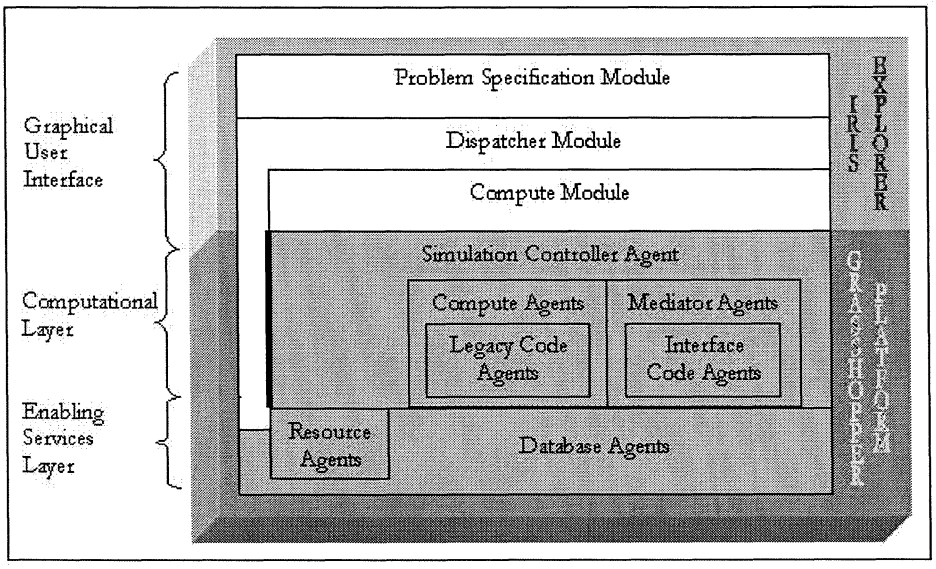

Fig. 10. The MPSE framework architecture.

User Interface Layer: As described in Sec. 4, the *Problem Specification (PS)* Module with the embedded TrueGrid tool is used to specify the root domain and its decomposition. The formatted output from this module is directed to the *Dispatcher Module.* The dispatcher distributes the partitioned data to the local databases of selected hosts on the available computational grid. It selects the physical host locations for each subdomain computational agent, using information provided by the resource agents and a set of allocation algorithms for optimizing network connectivity and machine load. The dispatcher has 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 generates a *host allocation table* as its output. The dispatcher module may also be wired in an Explorer map for other data distribution tasks such as a distributed, collaborative solution analysis session. The output from the dispatcher module is directed to the *Compute Module (CM).* The CM chooses the appropriate mediator agents from the library of MPSE mediators, based on the solver interface data requirements of the selected simulators on each domain. The CM launches the simulation and controls the execution of the compute and mediator agents. It monitors the simulation process until the user-specified stopping condition is reached. The output of the CM is the simulation problem solution. Figure 5 illustrates the top level user interface.

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 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 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 accomplishes 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 informs the other relevant compute agents and mediator agents of the migration. However, since the Grasshopper environment supports location transparent communication for its

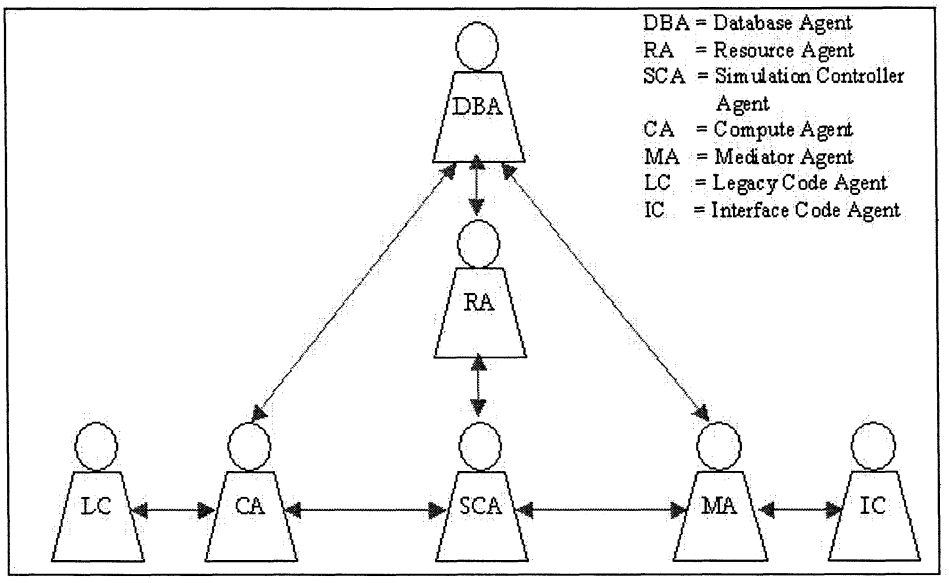

Fig. 11. PSE framework agent interactions.

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 for the compute agent are encapsulated within a *Legacy Code* (LC) *Agent.* The actual legacy codes for the mediator agent are encapsulated within an *Interface Code* (IC) *Agent.* This second tier of wrappers exists transparently within the PSE framework. Thus, all other agents in the framework interact solely with the compute and mediator agents and not the LC and IC agents. The LC and IC agents communicate only with their corresponding compute and mediator agents. The possible agent interactions within the PSE framework are schematically depicted in Fig. 16. 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 above). For clarity, we 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.

The LC and IC agent availability on the computational grid hosts is recorded as part of the resource information in the PSE framework. Thus, the LC and IC agent locations are considered by the allocation algorithms of the dispatcher module when assigning the partitioned domains to the computational grid hosts. This information also 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.

6. Simulation Results

This section reports on the results of two prototype simulations for a functioning gas turbine engine. The first prototype couples two principal sections of a compressor: the stator and the rotor. This prototype problem is simulated by executing ALE-3D legacy code on both subdomains. The second prototype couples the combustor and stator, applying different legacy codes (ALE-3D and KIVA-3V) to the two subdomains. In both prototypes, the common boundaries (Fig. 12) of the subdomains are treated with a mediator which communicates with the two executing simulators along the interface.

6.1. *The compressor prototype*

Initially, the implicit version of the ALE-3D legacy code is used to simulate both subdomains of the compressor prototype. Figure 12 shows the (stationary) stator and the (rotating) rotor, with the working area of the mediator marked on the interface annulus. The mediator requires specialized code to handle the simple but necessary computations involving interface data received from both simulations. Outflow data from the stator and inflow data from the rotor are processed by the mediator and the computed values are passed back to the two simulators. The processing is done once for each time step of the engine simulation. At simulation startup, an initial guess for the flow field variables is specified. This guess is not compatible with the boundary conditions of the problem (e.g. the pressure at the domain exit, solid surfaces within the domain). As the simulation marches forward in time and as the flow moves through the domain, the flow variables adjust

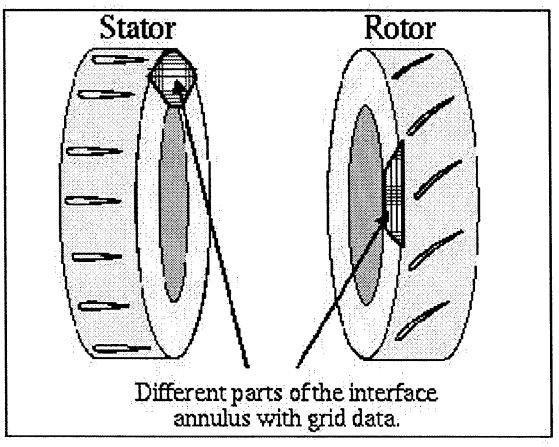

Fig. 12. The mediator work area is the annulus on the boundaries of the stator and rotor domains. The mediator resolves grid and solution value differences between the two simulators.

themselves to the boundary conditions so that eventually the flow field reaches a valid solution. This solution could include some periodic unsteadiness due to the relative rotation of one blade row with respect to another. Thus, the solution is usually referred to as an *unsteady steady state.* During the simulation, the mass flow rate through each domain is calculated and stored at regular time intervals. This mass flow rate is averaged over time T and 2T, where T is the period of unsteadiness in the flow (e.g. due to rotation of blade rows). If the average mass flow rate averaged over T is the same as that averaged over 2T, then the simulation has converged. The amount of time required to obtain an unsteady steady state is generally not known a-priori, but a good guess for this time can be made by an experienced user. In the prototype, the time required for the simulation to reach an unsteady steady state is input directly by the user into each legacy code; this time is the same for all codes involved. Legacy codes' iterations stop when this time is reached.

The goal of the first prototype is to achieve solution results across both subdomains with the desired level of accuracy, in particular along the interface boundary. Passing the correct interface information between the simulations required significant changes to the original mediator code, resulting in a decision to construct mediators specialized for the MPSE and for the solvers, Sec. 6.4 discusses the resolution of the issue of building custom mediators. The remainder of this section assumes that customized code is added ro the kernel mediator code when required by the solvers of the subdomains involved in the mediation process.

Since the *implicit mediator* (used with the implicit version of the ALE-3D code for the combustor prototype) processes data from two different simulations, it must resolve all issues related to differences in their solution methods and data requirements. The mediator receives an initial grid definition for the common boundary from both simulations, and must handle different grids on some portion of the shared surface, as shown in Fig. 12. In addition, although the stator grid remains unchanged throughout the simulation, the rotor grid is redefined at each time step according to the given rotation speed. This results in added computations for the mediator, since interporlation points are different at each time step and must be redefined. The solvers use values both at the grid nodes and at the centers of the finite element faces and, in general, the types of nodal and element-based values passed to the mediator may be different for different simulators. In the prototype, nodal values for the stationary domain are velocities in each direction. Element-based values are pressure (from the rotor), and density and energy (from the stator). Using periodicity and interpolation, the mediator computes velocity at the rotor nodes, density and energy at the centers of the element faces for the rotor, and pressure at the center of the stator element faces. The computed values are sent to the appropriate simulations. For the interpolation, the four closest points on the opposite section (defined by periodocity) are used for the multivariant interpolating polynomial defined by "de Boor and Ron's Least Polynomial Method[35]". For this interface, no relaxation method is needed since the two domains exchange values for different quantities.

Since the resulting solutions are not sufficiently accurate (see the implicit mediator solution in Fig. 13), a new mediator is designed to provide greater accuracy. The new mediator operates with the *explicit version* of ALE-3D, and differs from the implicit mediator in the types of data values that are communicated (input and output) and in the computations. The *explicit mediator* processes velocity at the stator nodes, force and mass on both stator and rotor nodes, and density and energy at the element face centers of the stator and rotor. Element volumes are also provided to the explicit mediator, since they are needed to compute weighted sums of values from both subdomains. The mediator computes density and energy on element faces, and acceleration (as a function of force and mass) on the nodes for both subdomains. Velocity is also computed for the rotor nodes. Periodicity is considered, and interpolation is used to obtain the desired results. The significantly improved results for pressure are shown in Fig. 13 (right part), while the mass flown convergence is presented in Fig. 14 (T here is 166.67 μsec, since there are 18 blades on each part of the compressor and the rotation rate is 20,000 rpm). The ALE-3D code was modified by adding several Fortran routines to output key data to the mediator three times for each time step, and to read in the mediator supplied data at these times.

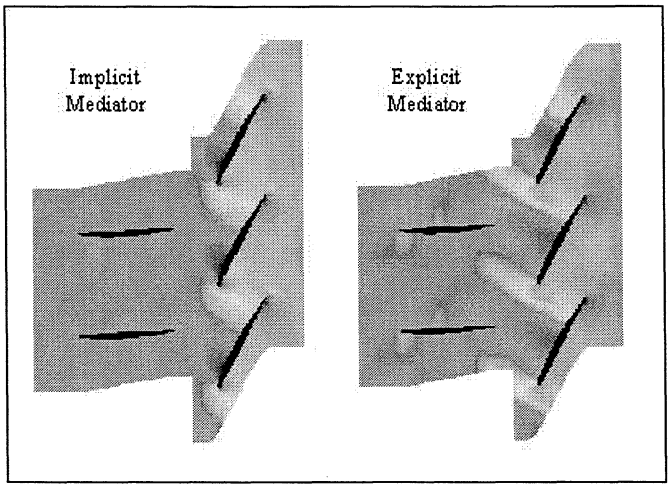

Fig. 13. Pressure computed on two subdomains by ALE-3D, with mediator interface relaxation on the common boundary. The explicit version of ALE-3D in conjunction with the explicit mediator produced good results.

6.2. *The combustor-stator prototype*

The combustor-stator prototype couples two different legacy code solvers. ALE-3D simulates the stator and KIVA-3V simulates the combustor. The *implicit mediator* originally designed for the compressor prototype is applied successfully to mediate the interface boundary, although some modifications are required to normalize data values since the unit systems of the two solvers are not the same. As with ALE-3D, the KIVA-3V legacy code is modified by adding two (Fortran) routines to write the common boundary data (pressure at the center of element faces) to a file for mediator input at each time step, and to read the values of velocity, density and energy which are computed and output by the mediator. Periodicity and space interpolation are used as before, and interpolation for time is not necessary since the two domains release their data at equal time steps. The criteria to get the solution (as in the compressor simulation) is the mass flow to reach a steady state. The history of convergence for this prototype is shown in Fig. 14 (right part) while the results of this prototype simulation are shown in Fig. 15. The GasTurbnLab Visualizer is used to produce the images shown.

6.3. *Prototype implementation*

Both of the legacy codes are written in Fortran, and the implicit and explicit mediators are implemented in C. The solvers and the mediator are compiled and executed on an SGI running IRIX64 version 6.5. For both prototypes, the three separate processes are coordinated by a Unix script, which stops the solver process at the end of each time step, calls the mediator, and resumes the solver execution when the mediator computations are finished.

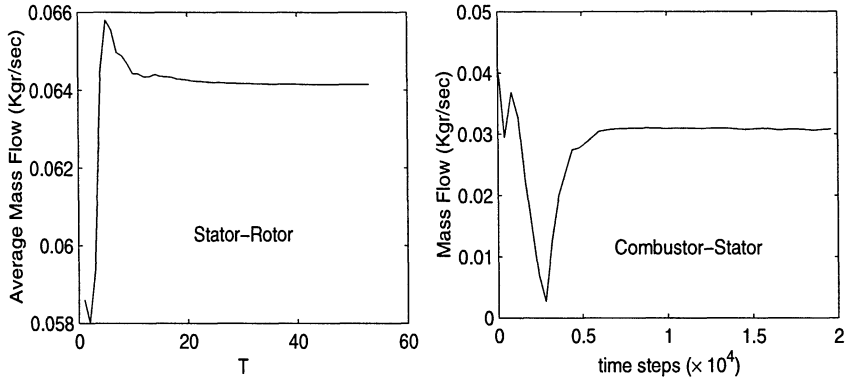

Fig. 14. Convergence history of mass flow for the compressor (left) and the combustor-stator (right) prototype.

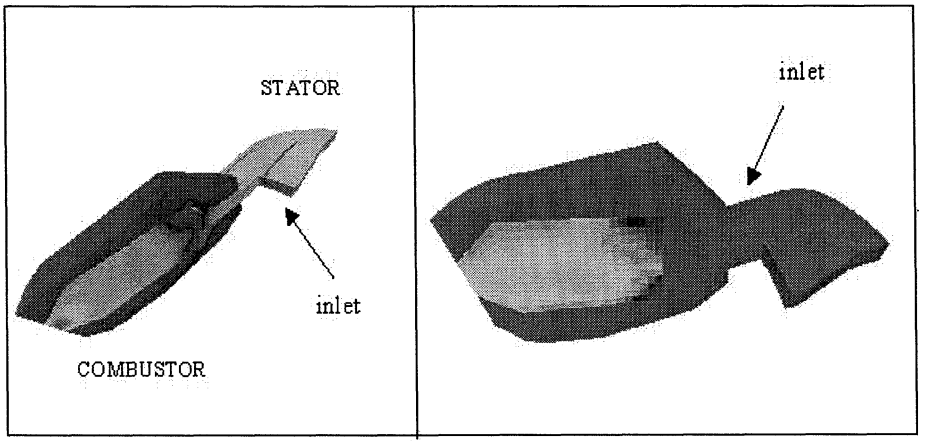

Fig. 15. Computational results for temperature produced from a mediated simulation by the solvers ALE-3D and KIVA-3V, communicating through the mediator at the combustor inlet. To the left is velocity, to the right, energy.

The "communication" between the three processes is achieved by writing/reading data to specific files The legacy code requires minimal changes, with two additional Fortran routine inserted for mediator input and output values, and a one line insertion for the call to the Unix script. Figure 16 shows the agent implementation of the compressor simulation. In the prototype implementation, the Simulation Control Agent is replaced by the Unix script, and the Control and Mediator Agents are empty wrappers for the legacy and mediator codes. The Grasshopper agent implementation is currently underway.

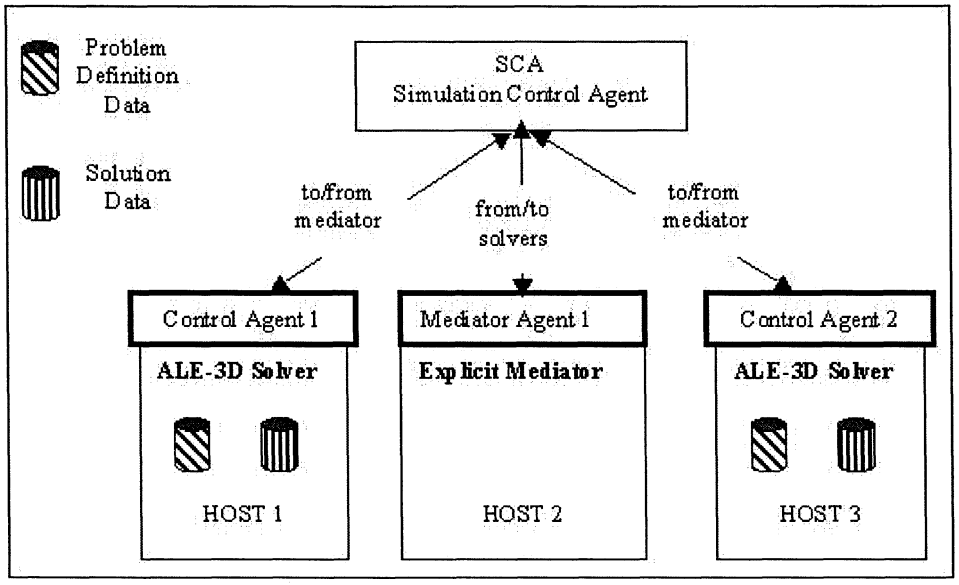

Fig. 16. Solvers and mediator interaction in the compressor prototype simulation. The information passing between the control and mediator agents is the interface data for force, mass, velocity and element volumes.

6.4. *Prototype issues*

A major issue was raised during the prototype implementation regarding legacy code modification (for mediator interface processing) vs. MPSE-specific, solver-specific customization of each mediator. The interface processing required to compute mediated values may be complicated and messy. In addition, the types of data values and the types of computations are specific to the simulation codes which are mediated. During the building of the first prototype, the issue of which code (legacy vs. mediator) is responsible for the computations was raised. Since one of the objectives of MPSE legacy code integration is that the legacy code should not be modified in any significant way, this forces the mediators to be both MPSE and solver specific, resulting in a case-by-case determination of customized code to be added to the mediator kernel code. Thus, MPSEs will have a library of custom mediators, and the specific mediator used to mediate two solver agents will be determined by the Simulation Control Agent based on the two legacy solvers involved and the kind of calculations required for the specific instance of domain-to-domain communication.

In order to allow the mediator kernel to be general, a Java/C-agent wrapper for the mediator can be implemented (Fig. 17) to handle case-specific data and computations. This defines a clean, layered architecture for the mediator agents. The innermost core consists of general mediator code, and each layer around it introduces additional specialization.

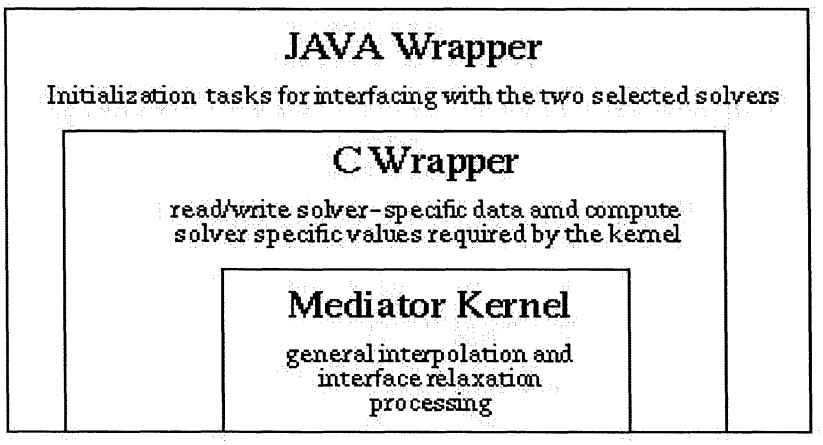

Fig. 17. Layered architecture for the mediator agent.

7. Conclusion

In summary, we present 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 use a layered architecture for the framework to incorporate 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 are important goals 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 interface 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 Sec. 4. A suitable load balancing algorithm can be implemented with 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, or 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.

References

1. A. C. Catlin, M. G. Gaitatzes, E. N. Houstis, Z. Ma, S. Markus, J. R. Rice, N. H. Wang, and S. Weerawarana, "The SoftLab experience: Building virtual laboratories for computational science", CSD-TR-95-041, Dept. of Computer Sciences, Purdue Univ., 1995.
2. 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 (1994).
3. 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.
4. T. Drashansky, S. Weerawarana, A. Joshi, R. Weerasinghe, and E. N. Houstis, "Software architecture of ubiquitous scientific computing environments for mobile platforms", CSD-TR-95-032, Dept. of Computer Sciences, Purdue Univ., 1995.
5. S. Fleeter, E. N. Houstis, J. R. Rice, and C. Zhou, "GasTurbnLab Design", Department of Computer Science, Purdue University, CSD-TR-99-002, January 1999.
6. S. Fleeter, E. N. Houstis, J. R. Rice, and C. Zhou, "Gas Turbine Engine Compressor - Combustor Dynamics Simulation Design", Department of Computer Science, Purdue University, CSD-TR-99-006, February 1999.
7. 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.
8. E. Gallopoulos, E. N. Houstis, and J. R. Rice, "Computer as thinker/doer: Problem solving environments for computational science", *IEEE Computing in Science and Engineering* **1**, 11 (1994).
9. 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**, 479 (1994).
10. E. N. Houstis, J. R. Rice, and R. Vichnevetsky (editors), *Intelligent Mathematical Software Systems* (North Holland, Amsterdam, 1990).

11. 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*, ed. P. Gaffney and E. Houstis (North-Holland, Amsterdam, 1992), pp. 229-241.
12. E. N. Houstis and J. R. Rice, "The architecture of PDE solving systems", in *Computer Methods for Partial Differential Equations VII*, ed. R. Vichnevetsky (IMACS, New Brunswick, NJ, 1992), pp. 363-370.
13. E. N. Houstis, J. R. Rice, and S. Weerawarana, "A software platform for integrating symbolic computation with a PDE solving environment", in *Proceedings of 14th IMACS World Congress* (IMACS 1, 1994), pp. 482-485.
14. Industrial Research Institute, *Proceedings: Roundtable Meeting on Reducing R&D Cycle Time* (Industrial Research Institute, Washington DC, 1992).
15. W. R. Johnson, Jr., "Anything, Anytime, Anywhere: The future of networking in Technology 2001: The future of Computing and Communications", ed. D. Leebaert (MIT Press, Cambridge, MA, 1992).
16. A. Joshi, S. Weerawarana, E. N. Houstis, J. R. Rice, and 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.
17. S. B. Kim, S. Markus, N. E. Houstis, E. N. Houstis, A. C. Catlin, and P. Wu, "Parallel methodologies for 'legacy' scientific software", in *Proceedings of Intel Supercomputer Users Group* (1995).
18. 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), pp. 381-387.
19. H. S. McFaddin and J. R. Rice, "Collaborating PDE solvers", in *Applied Numerical Mathematics* **10**, 279 (1992).
20. M. Mu and J. R. Rice, "Modeling with collaborating PDE solvers - Theory and practice", in *Contemporary Mathematics* **180**, 427 (1994).
21. 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*, ed. D. Keyes *et al.* (SIAM Pubs., 1992), pp. 129-150.
22. J. R. Rice, "An agent based architecture for solving partial differential equations", in *SIAM News* **31**, No. 6, 1 (1998).
23. 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*, ed. Donald, Kapur and Mundy (Academic Press, 1992), pp. 303-321.
24. S. Weerawarana, E. N. Houstis, J. R. Rice, A. C. Catlin, C. L. Crabill, C. C. Chui, and S. Markus, "PDELab: An object-oriented framework for building problem solving environments for PDE based applications", in *Proceedings of 2nd Object-Oriented Numerics Conference*, ed. A. Vermeulen (RogueWare Software, Corvallis, OR, 1994), pp. 79-92.
25. S. Weerawarana, "Problem Solving Environments for Partial Differential Equation Based Systems", Ph.D. Thesis, Department of Computer Sciences, Purdue University, 1994.
26. 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", in *Enabling Technologies for Computational Science: Frameworks, Middleware and Environments*, ed. E. Houstis, S. Gallopoulos, J. Rice, and R. Brambley (Kluwer Press, 2000).

27. P. Wu, E. N. Houstis, and J. R. Rice, "EPPOD: A parallel problem solving environment for the electronic prototyping of physical objects design", in *Proceedings of DAGS '94 Symposium*, ed. F. Makedon (Dartmouth Inst. Adv. Grad. Studies, Dartmouth, NH, 1994), pp. 135-151.
28. P. Wu and Elias N. Houstis, "A parallel mesh generation and decomposition methodology", in *Proceedings of Mesh Generation Conference*, Albuquerque, Oct. 1994.
29. P. Wu, "Parallel shape optimization", in *Proceedings of International Conference on Parallel Algorithms*, Wuhan-China, 1995.
30. D. M. Young and B. R. Vona, "On the use of rational iterative methods for solving large linear system", in *Applied Numerical Mathematics* **10**, 261 (1992).
31. Sun Microsystems, The Network is the Computer, Trademark.
32. The Grasshopper Agent Platform, IKV++ GmbH, Kurfurstendamm 173-174, D-10707 Berlin, Germany. http://www.ikv.de.
33. IRIS Explorer Toolkit, IRIS Explorer Center (North America), 1400 Opus Place, Suite 200, Downers Grove, IL 60515-5702, USA. http://www.nag.com/IEC.
34. TrueGrid, XYZ Scientific Applications Inc., USA. http://www.truegrid.com.
35. Carl de Boor and Amos Ron, "The least solution for the polynomial interpolation problems", *Mathematische Zeitschrift* **210** 705 (1992).