
ELSEVIER

Journal of Computational and Applied Mathematics 149 (2002) 83–100

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.com/locate/cam

GasTurbnLab: a multidisciplinary problem solving environment for gas turbine engine design on a network of nonhomogeneous machines☆

E.N. Houstis[a,b,c,*], A.C. Catlin[a], P. Tsompanopoulou[d], D. Gottfried[a], G. Balakrishnan[a], K. Su[a], J.R. Rice[a]

[a] *Department of Computer Sciences, Purdue University, West Lafayette, IN 47807, USA*
[b] *Department of Computer Engineering and Informatics, University of Patras, Patras, Greece*
[c] *Department of Computer Engineering and Communications, University of Thessaly, Volos, Greece*
[d] *Department of Informatics, University of Oslo, Oslo, Norway*

Received 31 October 2001; received in revised form 31 January 2002

Abstract

Gas turbine engines are very complex (with 20–40,000 parts) and have extreme operating conditions. The important physical phenomena take place on scales from 10–100 microns to meters. A complete and accurate dynamic simulation of an entire engine is enormously demanding. Designing a complex system, like a gas turbine engine, will require 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 describe the architecture of an agent-based software framework for the simulation of various aspects of a gas turbine engine, utilizing a "network" of collaborating numerical objects through a set of interfaces among the engine parts. Moreover, we present its implementation using the Grasshopper agent middleware and provide simulation results that show the feasibility of the computational paradigm implemented.

Keywords: Multidisciplinary problem solving environments; Mobile agents; Mathematical network model; Interface relaxation; Domain decomposition

☆ This work was partially supported by the US Department of Energy ASCI program contract LG-6982 and NSF grants 0075506-EIA, DMI-0122214.

* Corresponding author. Department of Computer Sciences, Purdue University, 47807 West Lafayette, IN, USA. Tel.: +1-765-494-6181.

E-mail address: enh@cs.purdue.edu (E.N. Houstis).

0377-0427/02/$ - see front matter
PII: S0377-0427(02)00522-8

1. Introduction

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 partial or ordinary differential (PDE or ODE) system with various formulations for the geometry, PDE, ODE, interface/boundary/linkage and constraint conditions in many different geometric regions [6,23]. 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 a new computational paradigm 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 [9]. 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 [2,12,15].

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 for the case of collaborating PDE models in [16–19,21,22]. The results obtained so far verify the feasibility and potential of network-based prototyping. Throughout this paper, we refer to the framework and software kernel for supporting the above computational scenario as the multidisciplinary problem solving environment (MPSE) for simulating multidisciplinary applications [4,9,11].

We believe that the software architecture of an MPSE can be effectively supported by the collaborating PDE solvers simulation approach [1,3,14,24] and be realized by an agent-oriented [5] paradigm. MPSEs must exploit and build on the new technologies of computing. This paper describes the implementation of an MPSE framework based on a commercial agent-based system for the gas turbine engine design which has all the characteristics of a multidisciplinary application. The paper is organized as follows. Section 2 describes the gas turbine engine as a multi-physics application. Section 3 presents the design and application infrastructure of the agent-based MPSE framework. Section 4 describes the implementation of the GasTurbnLab MPSE using this framework while in Section 5, we present results obtained by GasTurbnLab for four engine configurations and discuss computational issues such as convergence and time stepping selection, and quality of simulations. We conclude our discussion in Section 6 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.

2. The gas turbine engine multidisciplinary application

The gas turbine engine is an engineering triumph. It has more than 1300 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 10 g, with flow path pressures and temperatures to 40 atmospheres and 1400 F. 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. In this paper, we consider simulating the compressor–combustor–turbine coupling in a gas turbine engine [3]. For this, we plan to design and implement an agent-based MPSE, referred to as GasTurbnLab, to study complex physical phenomena such as stall, surge and turbine blade fatigue. The software infrastructure supporting CFD and combustion simulations in the prototype GasTurbnLab MPSE consists of ALE3D and KIVA-3V legacy FORTRAN codes. The hardware infrastructure for these simulations and the implementation of GasTurbnLab MPSE is a grid 96 heterogeneous CPUs organized in three homogeneous Intel PC clusters running Solaris. For the gas turbine engine design simulations, we use a collaborating PDE solvers paradigm [2,8,10] implemented by the Grasshopper Agent Platform [7] which is mobile agent system interoperability facilities specification (MASIF). It can be easily extended to any grid environment.

3. 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 including gas turbine engine. 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 [16,18]. 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.

3.1. 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

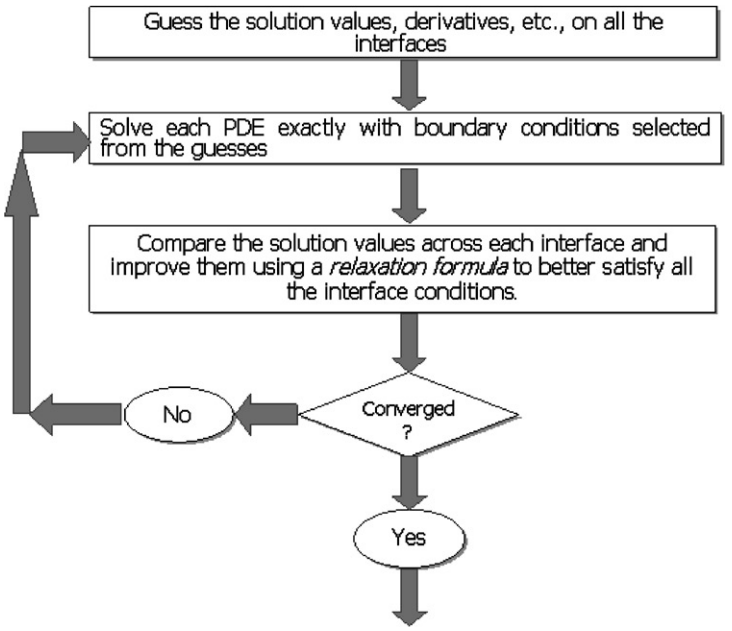

Fig. 1. The interface relaxation iteration.

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 overheard 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 Fig. 1. 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. 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 [7]. A brief discussion of Grasshopper is given first, since a description of its functionality is essential for describing the MPSE agent-based framework.

3.2. 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

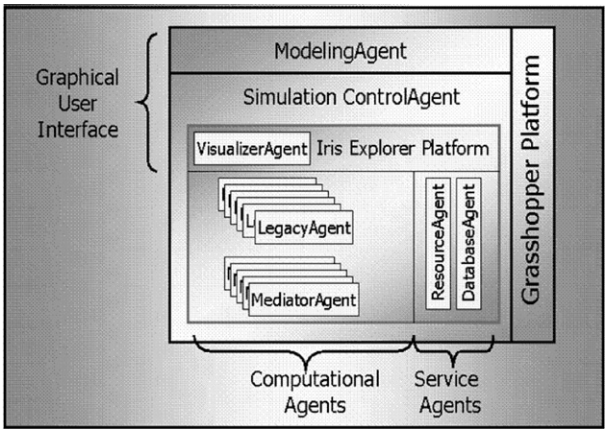

Fig. 2. The agent-based design of the MPSE. The SimulationControlAgent is the client which controls the entire simulation process. Legacy and mediator codes (generally Fortran or C programs) are transformed into server agents.

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 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.

3.3. 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 (Fig. 2): 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. 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 [20]), 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 [13]. 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.

3.4. Implementation of a LegacyCodeAgent

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. 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 (Fig. 3):

- 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.

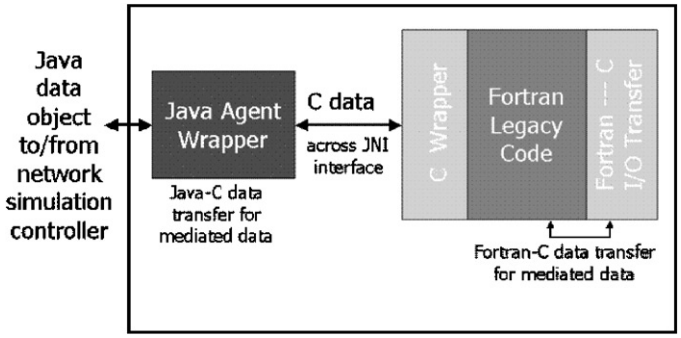

Fig. 3. Generic template procedure for embedding legacy Fortran code into a Java Grasshopper agent. This procedure transforms a Fortran executable into an LCA legacy library, which can be invoked and controlled by the LCA.

The 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.

3.5. Implementation of a MediatorAgent

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.

3.6. 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 VA when requested to do so. The entire simulation process begins and ends with the SCA.

4. GasTurbnLab: an agent-based MPSE for gas turbine engine design

We now present the implementation of GasTurbnLab assuming the framework described in the previous section and a simplified view of the turbine engine as it is illustrated in Fig. 4. We identify two existing legacy code PDE solvers to be encapsulated as LCA, and we write the customized interface mediation code for two MA to support the exchange of interface boundary data between the LCAs.

4.1. The GasTurbnLab LegacyCodeAgents

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

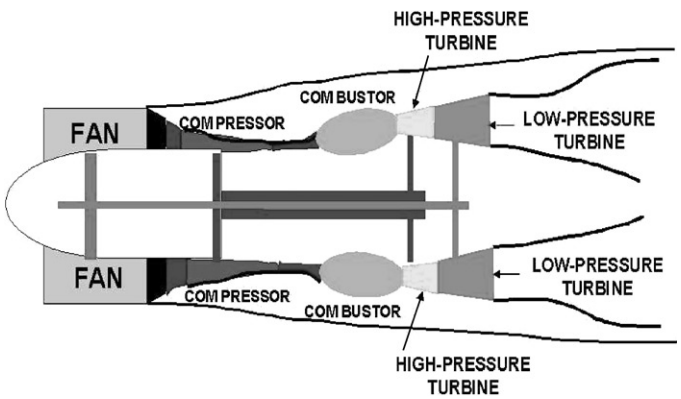

Fig. 4. A simplified view of a turbine engine. The fan and the compressor consist of many rows of stators and rotors, ending with a stator in front of the combustor.

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 5. Simulations for more complicated configurations that answer questions about stall, surge and fatigue are also described; these simulations are currently underway.

4.2. The GasTurbnLab MediatorAgents

Mediation code is MPSE specific code that handles the exchange of boundary data on the interface between two domains. Two MAs, a basic mediator (BasicMA) and an advanced mediator (AdvMA), 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 AdvMA provides a more accurate solution than the 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 either inflow or outflow boundary is open to the atmosphere, while the 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.

4.3. 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.

5. GasTurbnLab: simulation results

In this section, we present some numerical results to indicate (a) the user specified parameters involved, (b) the convergence criteria of the *interface relaxation method* used, and (c) the quality of results obtained. The results are based on four configurations of engine parts consisting of stators, rotors and combustors that have been simulated. Results on more complex turbine engine configurations and the performance evaluation of the GasTurbnLab MPSE will be presented elsewhere. The engine configurations considered here explore 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 mediation includes resolution of nonmatching discrete interfaces using interpolation and implementation of the relaxation method specified in Section 3. The time stepping of the

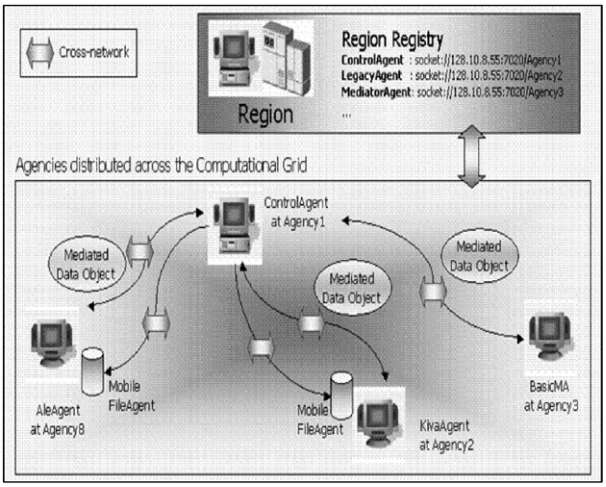

Fig. 5. Sample simulation scenario from the GasTurbnLab MPSE. The scenario shows a stator–combustor two domain simulation, mediated at the outflow boundary of the stator and the inflow boundary of the combustor by the BasicMA mediator agent. They communicate boundary interface information using the SCA as an intermediary. Users interact primarily with the Modeling and VisualizerAgents. All agents are built and executed within the Grasshopper environment.

ALE3D- and KIVA-based agents must be specified by the user so that these two codes communicate at the same simulation time. The nature of the computations involved in these two codes requires multiple steps of ALE3D for a single KIVA step. The quality of convergence for each engine configuration is assessed by criteria associated with each legacy code. In the case of ALE3D, we monitor the absolute change of the mass flow rate averaged over a period. For the KIVA code, we monitor the relative change of velocity, pressure, kinetic energy, dissipation rate, and chemical species. In all computations presented here, each agent code was executed on a single CPU. It's worth noticing that GasTurbnLab can run on a network of heterogeneous processors that support the Grasshopper middleware.

The first configuration (case 1) connects 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 (see the agent diagram in Fig. 5). This simulation was run for several variations of geometry, initial pressure conditions and number of iterations. Results of the stator–combustor simulations are shown in Figs. 6 and 7. Fig. 6 shows the mass flow rate at the inflow to the ALE3D domain. This mass flow rate has reached a steady-state value, indicating a converged aerodynamic solution. Fig. 7 shows the temperature profile within the combustor, and reveals the presence of a flame.

The second configuration (case 2) connects an upstream stator to a downstream rotor and is simulated using two AleAgents. The interface boundary was mediated by the AdvMA, and involves the exchange of force, flux and velocity data, with three mediator exchanges per time step. This simulation ran for 16,000 iterations with a time step of 0.5 μs. The resulting mass flow rate, Fig. 8, has a periodic character due to the relative motion of the rotor and stator for this highly loaded configuration. The measure of convergence in this case is the mass flow rate averaged over a period of unsteadiness. The time-averaged mass flow rate changes by less one-tenth of a percent for the

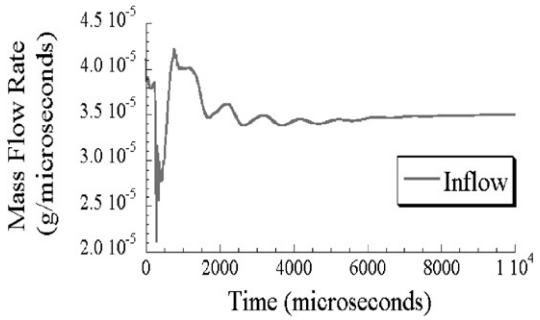

Fig. 6. The mass flow rate at the stator inflow moves towards a steady-state value after 1000 KIVA time steps (10,000 corresponding ALE3D time steps) in the stator–combustor simulation (case 1).

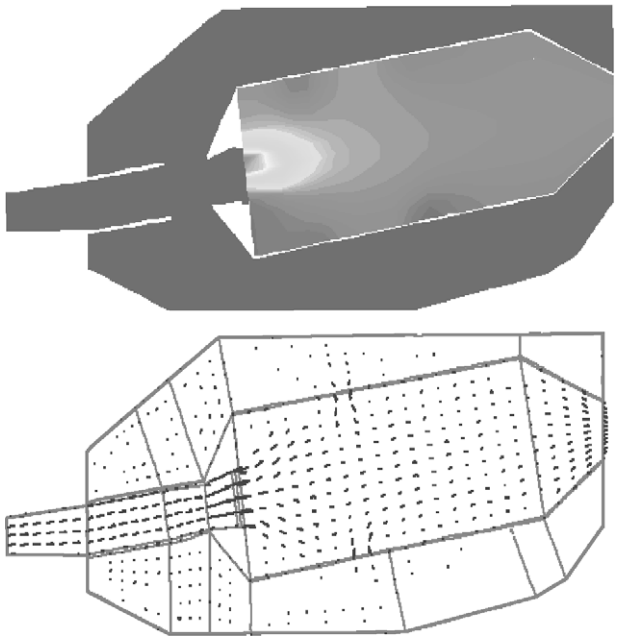

Fig. 7. Combustor plots which show the temperature contour indicating an ignition process in the beginning of combustion (top) and the velocity vector field (bottom). The results are from a case 1.

last two cycles, indicating a converged solution. The final mass flow was 5.69e-5 g/μs at stator inflow and 5.71e-5 g/μs at the rotor outflow. The total pressure ratio was 1.587. The axial velocity is depicted by the color contour plot in Fig. 10. It shows a shock emanating from the rotor and impacting the upstream stator. The presence of the shock indicates the flow field is transonic in the rotor frame of reference.

The engine configuration of case 3 is depicted in Fig. 9. It contains three AleAgents, simulating the stator, rotor, and stator passages upstream of the combustor, and one KivaAgent simulating the

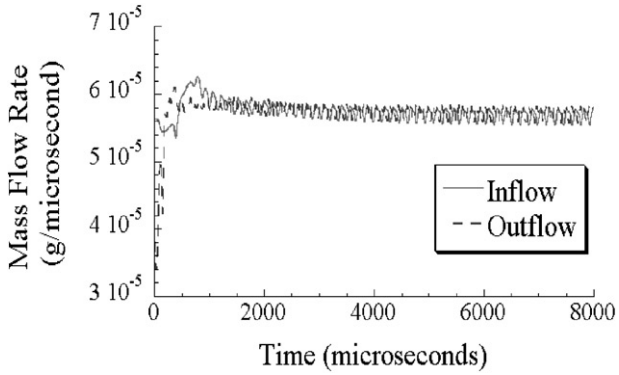

Fig. 8. Mass flow rate time history for the inflow of the stator and outflow of the rotor for case 2.

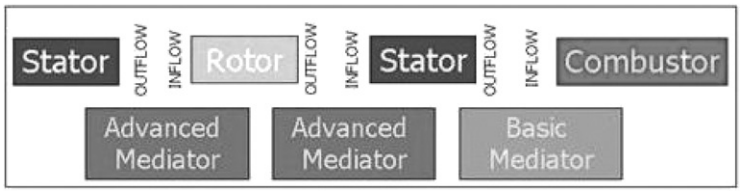

Fig. 9. A block diagram of case 3 engine configuration.

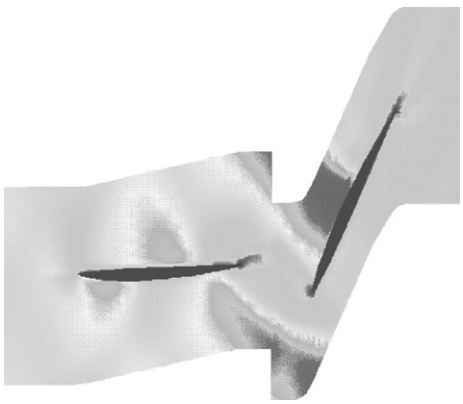

Fig. 10. Axial velocity plot for case 2. Maximum velocity (red) is 180 m/s; minimum velocity (blue) is 0 m/s.

combustor flow field. The interface boundary between the first stator (also called the IGV) and the rotor and between the rotor and the downstream stator are mediated by the AdvMA. The interface boundary between the downstream stator and combustor is mediated by the BasicMA. This case serves as the first test case for a rotating ALE3D domain upstream of a stator, but also as the first case using the two types of mediator in one simulation. Fig. 11 shows the mass flow rate time history for the inflow of the first ALE3D domain and the outflow of the third ALE3D domain. After

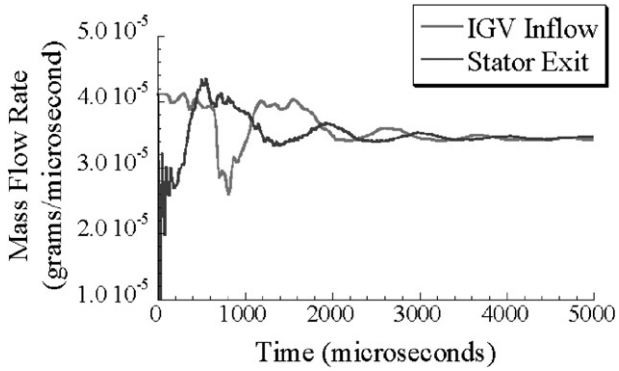

Fig. 11. Mass flow rate time history for the inflow of the IGV and outflow of the stator for case 3.

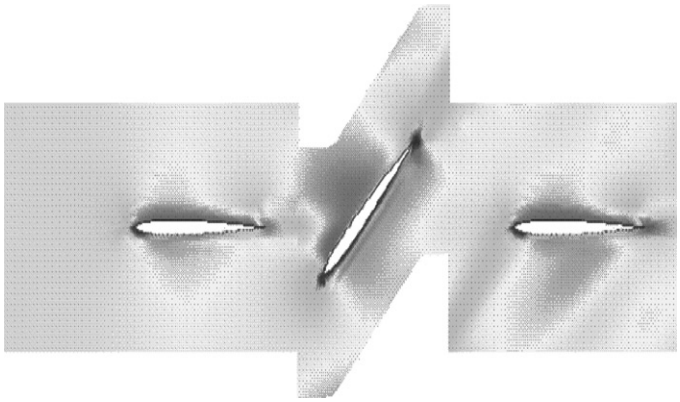

Fig. 12. Axial velocity plot for case 3. Maximum velocity (red) is 85 m/s; minimum velocity (blue) is 30 m/s.

5000 μs, the mass flow rate has reached a nearly constant value, indicating sufficient convergence. The final mass flow rate is 3.42e-5 g/μs. This case does not display a periodic unsteadiness in the mass flow rate because the blades were designed to have zero loading. Fig. 12 shows the axial velocity field for all three ALE3D domains. As evident in the figure, the airfoils are zero-camber airfoils having a NACA0008 profile. The flow conditions were specified such that each airfoil has zero incidence angle, and thus experiences zero loading. The rotor is at 45° stagger and is rotating at 5511 rpm. No shock waves are evident in this figure since the entire flow field is subsonic. As expected with the zero loading design, the total pressure ratio across the three ALE3D domains is 1.00. There appear to be wakes emanating from the aft portion of the airfoils. However, these are not true viscous wakes because viscous effects are not modeled in the ALE3D domains. These apparent wakes are due to small numerical losses from the airfoil surface boundary conditions being propagated downstream as velocity deficits. Fig. 13 shows the velocity vectors for the combustor flow field. Due to low activation energy and low stagnation pressure in the combustor, the flame did not start for this simulation.

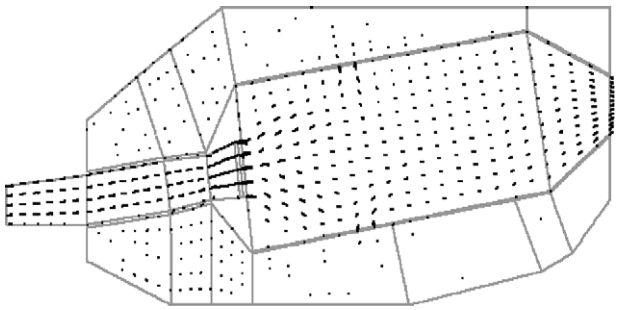

Fig. 13. Velocity vector plot for combustor flow field for case 3.

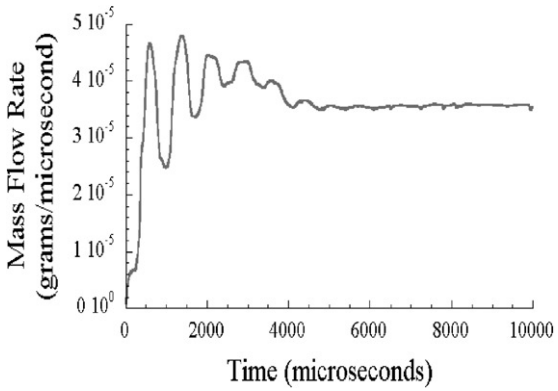

Fig. 14. Mass flow rate at outflow of ALE3D domain for case 4.

Case 4 has one ALE3D domain and one KIVA domain. The ALE3D domain is a stator downstream of the combustor. This is the first test case of the BasicMA with the ALE3D domain downstream. The ALE3D stator profile is again a zero-camber NACA0008 blade designed to have zero loading. Fig. 14 shows that after 10 ms of simulation the mass flow rate has reached a constant value of 4.00e-5 g/μs. Fig. 15 shows the axial velocity color contour plot of the ALE3D domain. Fig. 16 shows the temperature contours for the combustor. Ignition has just begun in this simulation, so the flame has not yet reached its steady-state temperature.

These four test cases demonstrate that the various interface boundary conditions, including the two kinds of mediators, are working properly.

6. Conclusion

In summary, we have described an agent-based framework for building multidisciplinary problem solving environments for complex, large-scale simulations of physical artifacts consisting of numerous parts with physical and geometric interfaces. The design is based on a "network" of

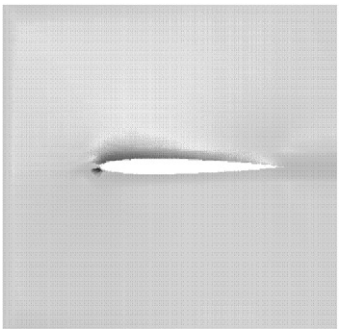

Fig. 15. Axial velocity plot for case 4. Maximum velocity (red) is 114 m/s; minimum velocity (blue) is −40 m/s.

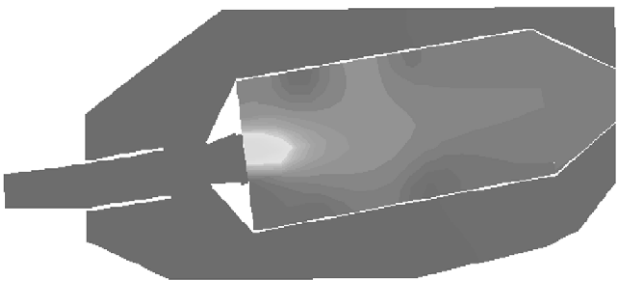

Fig. 16. Temperature contour plot, for case 4, showing ignition is just beginning.

collaborating numerical objects through the interfaces among the artifact parts. 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 utilizing the mobile functionality of the grasshopper agents. 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.

References

[1] L. Boloni, D.C. Marinescu, J.R. Rice, P. Tsompanopoulou, E.A. Vavalis, Agent based scientific simulation and modeling, Concurrency: Practice Experience 12 (2000) 845–861.

[2] T.T. Drashansky, E.N. Houstis, N. Ramakrishnan, J.R. Rice, Networked agents for scientific computing, Assoc. Comput. Mach. 42 (3) (1999) 48–54.

[3] S. Fleeter, E. Houstis, J. Rice, C. Zhou, A. Catlin, GasTurbnLab: a problem solving environment for simulating gas turbines, Proceedings of the 16th IMACS World Congress, August 2000, No. 104-5, 5pp.

[4] E. Gallopoulos, E.N. Houstis, J.R. Rice, Computer as thinker/doer: problem solving environments for computational science, IEEE Comput. Sci. Eng. 1 (1994) 11–23.

[5] M.R. Genesereth, An agent-based framework for interoperability, in: J.M. Bradshaw (Ed.), Software Agents, MIT Press, Cambridge, MA, 1997, pp. 317–345.

[6] D.A. Gottfried, Simulation of fluid-structure interaction in turbomachines, Ph.D. Thesis, Mechanical Engineering Department of Purdue University, May 2000.

[7] The Grasshopper Agent Platform, IKV++ GmbH, Kurfurstendamm 173–174, D-10707 Berlin, Germany, http://www.ikv.de.

[8] E.N. Houstis, G. Balakrishnan, A. Catlin, N. Dhanjani, S. Lalis, M. Stamatogiannakis, J.R. Rice, C. Houstis, Network-based scientific computing, in: R.F. Boisvert, Ping Tak Peter Tang (Eds.), The Architecture of Scientific Software, Kluwer Academic Publishers, Norwell, MA, 2001, pp. 3–28.

[9] E.N. Houstis, A. Joshi, J.R. Rice, T. Drashansky, S. Weerawarana, Towards multidisciplinary problem solving environments, HPCU J 1 (1) (1997), http://www.hpcu.org/hpcunews/vol1/issue1.

[10] E.N. Houstis, J.R. Rice, Future problem solving environments for computational science, Math. Comput. Simulation 54 (4–5) (2000) 243–257.

[11] E.N. Houstis, J.R. Rice, S. Markus, S. Weerawarana, Network based scientific problem solving environments, HPCU J. 1 (1) (1997), http://www.hpcu.org/hpcunews/vol1/issue1.

[12] E.N. Houstis, J.R. Rice, N. Ramakrishnan, T. Drashansky, S. Weerawarana, A. Joshi, C.E. Houstis, Agent based systems to support multidisciplinary problem solving environments for computational science, in: M. Zelkowitz (Ed.), Advances in Computers, 1998.

[13] IRIS Explorer Toolkit, IRIS Explorer Center (North America), 1400 Opus Place, Suite 200, Downers Grove, IL 60515-5702, USA, http://www.nag.com/IEC.

[14] S. Markus, E.N. Houstis, A.C. Catlin, J.R. Rice, P. Tsompanopoulou, E. Vavalis, D. Gottfried, Ke Su, G. Balakrishnan, An agent-based netcentric framework for multidisciplinary problem solving environments (MPSE), Internat. J. Comput. Eng. Sci. 1 (2000) 33–60.

[15] H.S. McFaddin, J.R. Rice, Collaborating PDE solvers, Appl. Numer. Math. 10 (1992) 279–295.

[16] H.S. McFaddin, J.R. Rice, RELAX: a platform for software relaxation, in: E.N. Houstis, J.R. Rice, R. Vichnevetsky (Eds.) Expert Systems for Scientific Computing, North-Holland, Amsterdam, 1992, pp. 175–194.

[17] M. Mu, J.R. Rice, Modeling with collaborating PDE solvers—theory and practice, Contemp. Math. 180 (1994) 427–438.

[18] J.R. Rice, P. Tsompanopoulou, E.A. Vavalis, Interface relaxation methods for elliptic differential equations, Appl. Numer. Math. 32 (1999) 219–245.

[19] J.R. Rice, P. Tsompanopoulou, E.A. Vavalis, Fine tunning interface relaxation methods for elliptic differential equations, Appl. Numer. Math., to appear.

[20] TrueGrid, XYZ Scientific Applications Inc., USA, http://www.truegrid.com.

[21] P. Tsompanopoulou, Collaborative PDE solvers: theory and practice, Ph.D. Thesis, Mathematics Department, University of Crete, Greece, 2000.

[22] P. Tsompanopoulou, L. Boloni, D. Marinescu, J. Rice, The design of software agents for a network of PDE solvers, Proceedings of the Agents'99: Third International Conference on Autonomous Agents, ACM Press, New York, 1999.
[23] R.G. Voigt, Requirements for multidisciplinary design of aerospace vehicles on high performance computers, ICASE Report No. 89-70, September 1989, 9pp.
[24] P. Wu, E.N. Houstis, EPPOD: a problem solving environment for parallel electronic prototyping of physical object design, J. Parallel Distributed Comput. 42 (1997) 157–172.