2000

GasTurbnLab Simulation of Instability in a Gas Turbine

A. C. Catlin

S. Fleeter

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

John R. Rice
Purdue University, jrr@cs.purdue.edu

C. Zhou

Report Number:
00-020

GASTURBN LAB
SIMULATION OF INSTABILITY IN A GAS TURBINE

A.C. Catlin
S. Fleeter
E.N. Houstis
J.R. Rice
C. Zhou

CSD TR #00-020
November 2000

GasTurbnLab
SIMULATION OF INSTABILITY IN A GAS TURBINE

A.C. Catlin[1], S. Fleeter[2], E.N. Houstis[1], J.R. Rice[1], and C. Zhou[2]

November 27, 2000

Abstract

This document describes the design and implementation of the GasTurbnLab system. The goal is to incrementally build a simulator of gas turbines that exhibits the onset of instability (stall) as described in [4]. A brief overview of the project is given and then the Grasshopper agent system is reviewed. Then the GasTurbnLab design and implementation is described. This system is operational and initial results are discussed.

1. OVERVIEW

The goal is to have a gas turbine simulation that exhibits stall and is as simple as practical. Currently, we can simulate combinations of one rotor blade, one stator blade and a combustor. There are three principal challenges to implement GasTurbnLab: (1) To embed the legacy PDE solvers into a modern, agent based problem solving environment, (2) To develop interface conditions between blades and the combustor and then to create mediator agents that handle them, and (3) To manage the computations efficiently on our network-based computing environment.

High-performance networks have already become an integral part of today's computing environments. This new capability allows applications to evolve from static entities located and running on specific hosts, to application agents that are spread across Internet or Intranet based networks [3,5]. In a network-based environment, the fundamental research problem can be viewed as the development of an infrastructure that supports the construction of computing environments by dynamically composing application-specific agents, (including "legacy code" agents), that are identified and linked together at run-time in a secure and robust way. GasTurbnLab is implemented using the agent-based Grasshopper system [8].

GasTurbnLab is based on the design document [4]. We assume the reader is familiar with this report. Then we discuss the creation of network-based agents which are designed (1) to control the legacy and mediator codes, and (2) to manage the entire computational environment.

2. Design and Implementation of GasTurbnLab

We first give a short overview of Grasshopper, and then describe the design and implementation of the agent-based GasTurbnLab computing environment.

---

[1] Department of Computer Sciences, Purdue University.
[2] Department of Mechanical Engineering, Purdue University.

2.1 The Grasshopper Agent System

Grasshopper is an agent development platform that supports distributed agent -based applications. The platform provides a base for communications services, mobile computing power and dynamic information retrieval. It is essentially a mobile agent platform that is built on top of a distributed processing environment, integrating the traditional client-server paradigm and mobile agent technology. The primary feature of the Grasshopper platform is its location independent computing, driven by the ability to move agents between different systems. It is powerful tool that facilitates the creation of agents, transparently locating them and controlling their execution. These agent -based applications are interoperable with other agent systems that are MASIF (Mobile Agent System Interoperability Facilities) compliant. The Grasshopper distributed agent environment is composed of regions, 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 componen ts, their movements and transport. Interactions can be performed by CORBA, IOP, RMI or plain socket, (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 loads by migrating. Stationary agents are associate d with a particular location only and are incapable of migration.

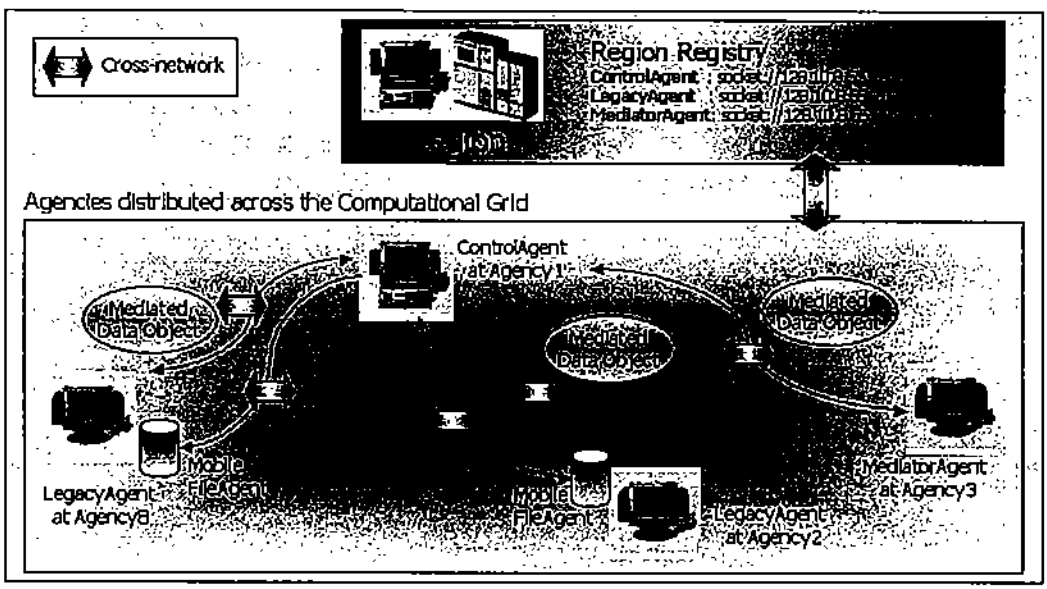

Figure 2.1. The agent-based network computing scenario for GasTurbnLab.

2.2 The GasTurbnLab Agent-based Design

The Grasshopper environment supports the traditio nal client -server structure. GasTurbnLab servers are Grasshopper agents, which provide computational service and data visualization. The GasTurbnLab client agent controls the entire simulation process by launching/terminating the server agents, managing th eir interactions and facilitating the cross - network asynchronous communication of computational and control data. GasTurbnLab is implemented using four classes of agents: the simulation control agent (SCA), the visualizer agent (VA), the legacy code agent s (LCA) and the mediator agents (MA). The SCA is the client which requests and manages the services of the remaining classes of agents. The VA is a server that receives solution data (velocity, speed, density, energy, pressure) for one or more engine parts, and renders it graphically using the Iris Explorer data visualization system [9]. The LCAs are the computational agents that simulate the engine; each agent encapsulates an established legacy code targeted for a specific engine part. The GasTurbnLab mod el requires the LCAs to communicate their 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 engine parts, each of which is simulated by one LCA. The GasTurbnLab SCA can handle any number of engine parts, i.e., it can control any number and type of communicating legacy code and mediator agents.

2.3 Implementation of the Legacy Code Agents

The initial ch allenge was to create a template procedure for embedding legacy Fortran codes into the server agent structure. The resulting LCAs could then exercise control over the Fortran code and enable data flow by starting up the legacy code, pausing after each iter ation to communicate the required boundary data to the SCA, receiving the mediated boundary data in return and continuing with the next iteration. Two legacy codes have been encapsulated as LCA agents: Ale3D, an advanced CFD code for simulating turbines [6] and Kiva, an advanced combustion simulation code [1].

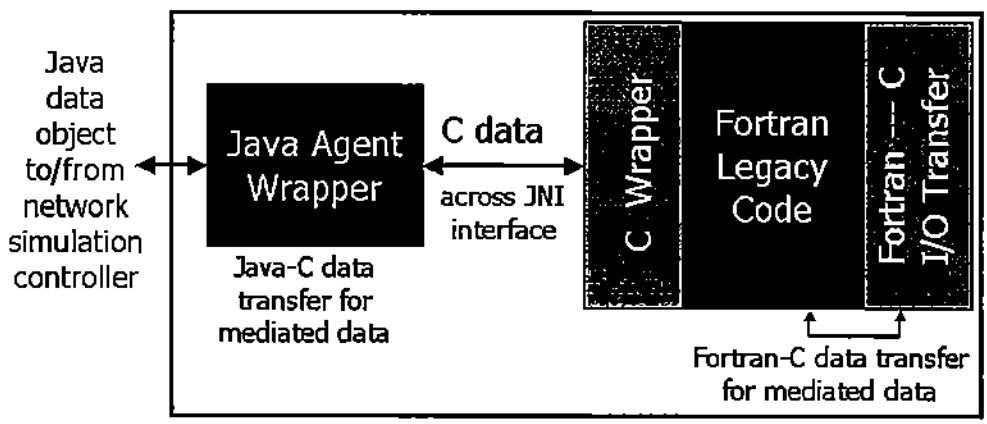

Figure 2.2. Implementation of a Legacy Code Agent.

The template procedure requires the legacy code, which generally starts out as a Fortran executable, to be transformed into a C -wrapped legacy code library as follows: (1) change the Fortran *main* to a subroutine, with command line arguments passed as parameters, (2) start the

Fortran subroutine as a thread from the C wrapper routine, (3) define C structures to hold the data representing boundary information, (4) write a C data transfer routine to copy boundary data back and forth between the C and Fortran data structures, (5) insert a call to the C data transfer routine in the Fortran iteration loop, and (6) insert cont rol code in the C wrapper to sleep/wake the C data transfer routine, thus effectively controlling the pause and restart of the Fortran iterations. The changes to the Fortran code are minimal. The C wrapper is defined with a JNI [11] interface to the Java agent code, and the C boundary data is passed up through the JNI interface parameters to the Java agent. The Fortran code and two C routines are compiled together as a *legacy library*, which is loaded into the Java LCA server when it is instantiated. The J ava legacy agent starts the C wrapper and waits for the JNI object containing the boundary data. When the object is received, the agent serializes it and communicates it to the SCA. The SCA returns a mediated data object to the agent, which copies it into a JNI object and passes it to the C wrapper. The SCA is also responsible for passing the legacy agent a termination signal. The LCA halts the Fortran code execution when the termination signal is received.

The GasTurbnLab project has successfully implement ed two legacy code agents: the AleAgent for simulating rotor and stator engine parts (also called *domains*) and the KivaAgent for simulating combustor domains.

2.3 Implementation of the Mediator Agents

Mediator agents are customized Java or C programs that are easily encapsulated as Java agents. The program is wrapped by agent code that handles the boundary data object communication between the MA and the SCA. In general, the mediator's interface relaxation code is heavily dependent on the legacy codes and data types that are being mediated. Two MAs, a basic mediator and an advanced mediator, have been developed for mediating the boundary data for the GasTurbnLab domain interfaces. Both MAs are capable of mediating interfaces for any combination of Ale and/or Kiva 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; f or unsteady problems, the advanced mediator provides a more accurate solution than the basic mediator. The basic MA was created as a simple solution to the boundary interface problem. Based on this experience, the advanced MA was created to provide a more accurate transfer of unsteady fluid mechanic phenomena between the domains.

The basic MA treats the boundary between domains in the same way inflow/outflow boundaries are treated when open to the atmosphere , i.e., when an inflow/outflow boundary is open to the atmosphere, atmospheric conditions are specified at the beginning of the simulation. The basic MA simply updates these atmospheric conditions after each time step based on the aerodynamic data from the adjacent domain. Except for the data being pa ssed to them, the inflow/outflow boundary algorithms are the same whether the boundary is connected to the open atmosphere or to another Ale or Kiva domain. The drawback of this technique is that the inflow/outflow boundary condition algorithms are inheren tly less accurate than the interior algorithms. The capability to use this mediator, however, has been maintained as a debugging and solution validation tool.

The advanced MA calculates elemental and nodal quantities at the interface between domains in ne arly the same way as the interior elemental and nodal quantities are calculated. When the advanced mediator is used, the inflow/outflow algorithms are bypassed and more accurate algorithms are employed. These more accurate algorithms are nearly identical t o the

interior algorithms. Specifically, there are two areas in which the advanced MA provides more accurate values at the interfaces between domains. First, the Ale 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 advanced MA maintains second -order upwinding at the interface between legacy codes; the basic MA does not. Second, the advanced MA more accurately handles the exit boundary of a domain du ring the Lagrange calculation when it is connected to another domain. Unlike the basic MA, the advanced mediator provides forces at the exit so that the acceleration of exit nodes are calculated the same way interior node accelerations are calculated.

2.5 The SCA and Simulation Control

The SCA is responsible for managing the simulation in accordance with the specifications provided by the user through the GasTurbnLab top level graphical user interface. Users select the engine parts (domains) to be simu lated, and the GUI displays available geometry and parameter files for the corresponding parts. Users choose an input files for each part, and these selections are passed to the SCA. The SCA then determines the software and hardware resources to be used in simulating the selected engine parts, including which LCAs and MAs are to be instantiated.

For each Ale domain, two input files are required. The first is the kernel file containing the geometry (i.e., the mesh) along with the aerodynamics conditions, s tructural properties, and other simulation control parameters. The second file is a control input file containing additional simulation parameters. The key parameters used by the SCA to govern the simulation are the number of iterations and the interface identifiers. The SCA uses the iteration number to determine when to terminate the agents and end the simulation. The interface identifiers, one for the inflow boundary and one for the outflow boundary, specify whether the boundary is open to the atmosphere, connected to another domain with the basic MA, or connected to another domain with the advanced MA. The Kiva domains require a geometry (mesh) file and a file of simulation control parameters. A similar interface mechanism determines how the boundaries a re to be mediated. The startup file data is passed to the legacy codes via mobile FileAgents instantiated by the SCA. All other agent interaction takes place via object communication between the stationary LCAs and MAs (across the computational grid), using the SCA as the intermediary.

2.5 GasTurbnLab Engine Simulations

Four two-domain prototype simulations have been implemented. The first and second prototypes are two-domain Ale stator-rotor simulations; the first prototype uses the basic MA and the second prototype uses the advanced MA. We assume a running Grasshopper region with 3 or 4 registered agencies on machines belonging to the GasTurbnLab computational grid (see Figure 2.1.) Two AleAgents, the MA and a VA are loaded into (any combination of) the agencies and automatically registered with the region. The SCA is loaded into an agency and started as a stator-rotor controller. The SCA first looks up the registry to determine the availability of the necessary servers. It then supplies the two AleAgents with grid and parameter information, and the server agents commence execution. After the encapsulated legacy code finishes one cycle of computation, the boundary data is copied upward from the Fortran code to the C -wrapper to the JNI object to the Java ag ent. The SCA receives this data from the AleAgents, then filters and serializes the data for the MA. The MA is called with this data and, after running the interface computations, it returns the mediated data to the SCA. The SCA restarts the AleAgents with the

mediated data. This process continues until the termination signal is given. The VA is launched with solution data from both domains, and users can interactively choose which domains and solution data to display. These two prototype simulations have run thousands of iterations and the solution output data for each has been verified.

The third and fourth prototypes are two-domain Ale-Kiva simulations involving the stator and combustor; the third prototype uses the basic MA and the fourth uses the advan ced MA. The SCA controls the simulation environment consisting of one AleAgent, one KivaAgent, the selected MA and a VA. Validation of the solution output data for these simulations is nearing completion.

2.6 The Final Simulation

The simulation environm ent implemented for the LegacyAgents, MediatorAgents and SCA has been designed to scale to any number of domains. Simulation control and data passing mechanisms are in place to support the final configuration. Mediator and legacy code modifications that ha ndle the interfaces imposed by the new configuration are currently underway.

Bibliography

1. Amsdem, A. A. (1997). KIVA-3V, A block-structured KIVA program for engines with vertical or canted valves. *Technical Report LA -13313-MS UC -1412*, Los Alamos National Laboratory.
2. Baker, M., Buyya, R., and Laforenza, D. (July 31 - August 6, 2000). The Grid: International efforts in global computing. *Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*. SSGRR '2000, to appear.
3. Fleeter, S., Houstis, E.N., Rice, J.R., Zhou, C., and Catlin, A.C. GasTurbnLab: A problem solving environment for simulating gas turbines, *Proc. 16th IMACS World Congress*, No. 104–5, 5 pages, August 2000.
4. Fleeter, S. Houst is, E.N., Rice, J.R., and Zhou C. Simulation of instability onset in a gasturbine. Comp. Sci. Tech. Rpt. 00-019, October 2000.
5. Foster, I. and Kesselman, C. (1999). The grid: The blueprint for a new computing infrastructure.
6. Hallquist, J. (1983). Theoretic al manual for dyna3d. *Technical Report UCID 19401* , Lawrence Livermore Laboratories.
7. Houstis, E.N., Catlin, A.C., Dhanjani, N. and Rice, J.R. The WebPDELab Server: A problem solving environment for partial differential equation applications, *Proc. of IMACS World Congress*, No. 104–3, 6 pages, August 2000.
8. IKV++ (2000). The grasshopper agent platform. Available via World Wide Web at http://www.ikv.de.
9. IRIS Explorer Center. (2000). Iris explorer toolkit. Available via World Wide Web at http://www.nag.com/IEC.
10. Markus, S., Houstis, E. N., Catlin, A. C., Rice, J. R., Tsompanopoulou, P., Vavalis, E., Gottfried, D., Su, K., and Balakrishnan, G. (2000). An agent -based netcentric framework for multidisciplinary problem solving environments (MPSE). *Computational Engineering Journal*, to appear.
11. Sun Microsystems (1999). The Java native interface. Sun Microsystems, Inc. Java Software.

12. Su, K. and Zhou. C. Numerical modeling of gas turbine combustor integrated with diffuser, *Proc. of NHTC'00*, No. 12229, 9 pages.