1999

GasTurbnLab PSE DESIGN

Sanford Fleeter

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

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

Chem Zhou

Report Number:

99-002

GasTurbnLab PSE DESIGN

Sanford Fleeter
Elias Houstis
John Rice
Chenn Zhou

CSD-TR #99-002
January 1999

GasTurbnLab PSE DESIGN

Sanford Fleeter, Elias Houstis, John Rice and Chenn Zhou*

February 2, 1999

Abstract

This document provides the initial design specifications of the problem solving environment (PSE) GasTurbnLab. The long term objective of GasTurbnLab is to evolve into a complete and versatile simulator for gas turbines to study their performance and operability. The shorter term objective of GasTurbnLab is to simulate the compressor-combustor-turbine coupling to study the mechanisms of stall, surge and turbine blade fatigue. Simultaneously, GasTurbnLab will explore new PSE methodologies such as agent based simulation, interface relaxation and geometry objects.

I OVERVIEW

The GasTurbnLab project is to develop the problem solving environment GasTurbnLab and to advance simulation technology. The gas turbine is an engineering triumph (with about 30,000 parts, 1,600 of which rotate very rapidly). It has extreme operating conditions (with 10-50,000 rpm, 1000-1500 °F temperatures, pressures of 20-50 atmospheres, and 5-10g loads). The important physical phenomena take place on scales from 10-1000 microns to meters. A complete and accurate dynamic simulation of an entire engine is enormously demanding; it is unlikely that the required computing power, simulation technology or software systems will be available in the next decade. See Figure 1 for a view of an aircraft gas turbine and further characteristics of its operation.

The primary goal of this research is to advance the state-of-the-art in very complex scientific simulations and their validation. The major challenges are in integration of science, engineering, and computation and then demonstrating that the results are reliable. The project's principal theme is to develop the *problem solving environment infrastructure for complex simulations* and its secondary themes are: (1) to *validate this technology for gas turbine dynamics*, (2) to understand the mechanisms and identify processors to the phenomena of *stall, surge and turbine blade failure*.

*This work is supported by the U.S. Department of Energy ASCI program, contract LG-6982. The authors addresses are: Fleeter, Dept. of Mechanical Engineering, West Lafayette, IN campus, Houstis and Rice, Dept. of Computer Sciences, West Lafayette IN campus, Zhou, Dept. of Mechanical Engineering, Calumet IN campus.

The two principal physical phenomena involved in this simulation are:

(a) **Unsteady Interaction of Gas Turbine Engine Fluid-Thermal-Structure Components.** The analysis of the unsteady operation of a gas turbine engine is an important aspect of propulsion system design. Namely, as efficiency requirements increase, stability margin's, i.e., rotating stall and surge margins, are necessarily reduced. To compensate, control systems are becoming more sophisticated including active performance seeking logic and neural networks. In addition, unsteady gas turbine engine operation often produces extreme loading for the turbomachinery blading, resulting in high cycle fatigue (HFC) failures. Thus, the accurate analysis of the peak blade row unsteady aerodynamic loading and vibratory responses is needed.

The numerical simulation of the effects during unsteady operation of a gas turbine engine, requires the accurate computation of coupled fluid, thermal and structural interactions. The GasTurbnLab simulator is intended to analyze the operability of a complete gas turbine engine. This necessarily includes the interaction of the turbomachinery and the combustor flow fields, i.e., a time-varying coupled analysis of the combustor and turbomachinery components, enabling the operability of the complete engine to be determined, including the response to the control system.

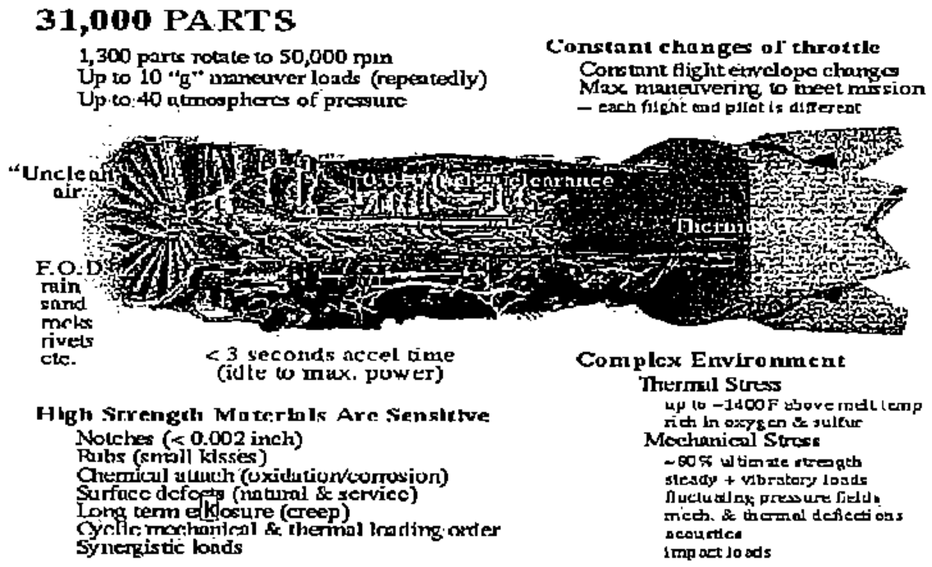

31,000 PARTS
1,300 parts rotate to 50,000 rpm
Up to 10 "g" maneuver loads (repeatedly)
Up to 40 atmospheres of pressure
Constant changes of throttle
Constant flight envelope changes
Max. maneuvering to meet mission
— each flight and pilot is different
"Unclean" air
Thermal
F.O.D. rain sand rocks rivets etc.
< 3 seconds accel time (idle to max. power)
Complex Environment
Thermal Stress
up to ~1400F above melt temp
rich in oxygen & sulfur
Mechanical Stress
~80% ultimate strength
steady + vibratory loads
fluctuating pressure fields
mech. & thermal deflections
acoustics
impact loads
High Strength Materials Are Sensitive
Notches (< 0.002 inch)
Rubs (small kisses)
Chemical attack (oxidation/corrosion)
Surface defects (natural & service)
Long term exposure (creep)
Cyclic mechanical & thermal loading order
Synergistic loads

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

(b) **Full Annular, Unsteady Flow in the Combustor.** CFD gas turbine combustor modeling has generally been limited to isolated parts of the combustion system. Most models include only the reacting flow inside the combustor liner with assumed profiles and flow spits at the various liner inlets. Carefully executed models of this type can provide valuable insight into mixing performance, pattern factor, emissions and combustion efficiency. A CFD calculation for the unsteady flow through a complete annulus combustor – from the compressor diffuser exit to the turbine inlet – is needed. The combustor configuration should be representative of a lean, low emission design. The model should include an airblast fuel nozzle, dome and liner walls with dilution holes and cooling louvers.

Consider a full-annular burner with time varying inlet circumferential distortions. GasTurbnLab should be able to simulate the time-varying circumferential combustor flow field including performance effects:

- Fuel schedule - mean flow design point.
- Potential for rich-hot combustor over part of annulus due to stall.
- Fuel flow pulsation effects on control.
- Feedback system control – as the combustor responds, it changes the back-pressure to the compressor that subsequently changes the compressor performance.

The initial GasTurbnLab PSE is to provide a base for future, more complex PSEs and to explore the characteristics and applicability of some of the new simulation methodology used. These include the use of:

- agent based, collaborating partial differential equation solvers [MuRice 95], [DHRR 99], [DraHo 97], [Rice 98],
- geometry object framework,
- interface relaxation [RVY 97], [RTV 99], and
- very high level software parts integration [Joshi 97], [HoRice 98a].

II POTENTIAL GasTurbnLab IMPLEMENTATION

The software context for the initial implementations of GasTurbnLab is as follows:

(a) There exists a large, versatile PSE called PELLPACK for partial differential equations (PDE) problems. It has about 2 million lines of code and encapsulates 15-20 PDE solving systems, plus many supporting software tools.

(b) There exists several large CFD codes targeted to gas turbine simulation. ALE3D with 200,000 lines of code is one of the more advanced and is being used at Purdue.

(c) There exists several large combustion simulation codes. KIVA with 50,000 lines of code is one of the more advanced and is being used at Purdue.

(d) No combination of these or any other codes solves all the problems that arise in gas turbine simulation.

II.1 GasTurbnLab Objectives

There are four classes of studies that the GasTurbnLab simulator could be used for and covering all of them impacts the GasTurbnLab design greatly. These objectives are:

Study #1: Design New Gas Turbines. One starts with a clean sheet of paper to design a new engine.

Study #2: Performance of an Existing Turbine. One has a design and wants to know how it performs in various situations. There are four operating regimes of primary interest:

* Steady or unsteady operation.
* On-Design or Off-Design operation.

Most existing gas turbine simulation efforts focus on steady, on-design studies.

Study #3: Validation of Computational Models. GasTurbnLab uses both physical models (mathematical equations) and numerical models (discretization methods). One wants to know if the models are valid.

Study #4: Performance of the Computation. One wants to know how the simulation speed depends on the machines, the models, and the software.

And, of course, GasTurbnLab should be reasonably economical to use and create! The design presented here is focused on objectives #2 and #3 with unsteady, off-design operations being essential (since the phenomena of the project's primary interest occur there).

II.2 Four Alternative Implementation Approaches

We list four alternatives for the initial GasTurbnLab implementation along with their drawbacks.

Alternative #1: Enlarge PELLPACK. We would wrap PELLPACK, KIVA and ALE3D with a new graphical user interface (GUI). We have already created one PSE this way and adding two more PDE solvers to PELLPACK should not be a big problem. However, (i) having 1.5 million surplus lines of code seems excessive, (ii) PELLPACK is structured to solve only a single PDE at a time. So computational efficiency and user friendliness are sure to suffer while an already complex code becomes more complex.

Alternative #2: All new code. Write all new software as no existing software (or combinations of existing software) is satisfactory. This might produce the (nearly) perfect PSE for gas turbines, but it would take many years and cost many 100's of millions of dollars.

Alternative #3: All new PSE with existing solvers. One could take the Navier-Stokes solver from PELLPACK plus ALE3D and KIVA as the PDE solvers and just redo the other parts of the PSE. One can hope that these solvers will evolve into complete 3D, full physics codes that are scalable, portable, reliable, accurate and validated solvers. Then the new PSE could make the required simulations. However, (i) the PDE solvers are not the "big" part of the PSE code; the GUI, AI support, data warehousing, etc., are large and difficult to do well. Figure 2 shows the structure of PELLPACK and one sees that the solvers are tucked away in two of the five boxes on the third level of the system. (ii) It is not clear what the final form of these solvers will take. One could be continually redoing the PSE as these solvers evolve. (iii) The parallelization of these three codes (or any similar codes) is likely to proceed slowly.

Alternative #4: Rebuild PELLPACK with ALE3D+KIVA. This would pare down PELLPACK by removing chunks of code not needed in GasTurbnLab. Then a gas turbine GUI would be added to PELLPACK, reusing much of the current GUI. However, (i) PELLPACK still solves only one PDE at a time, and (ii) the compromises made will probably impact the user negatively in some ways.

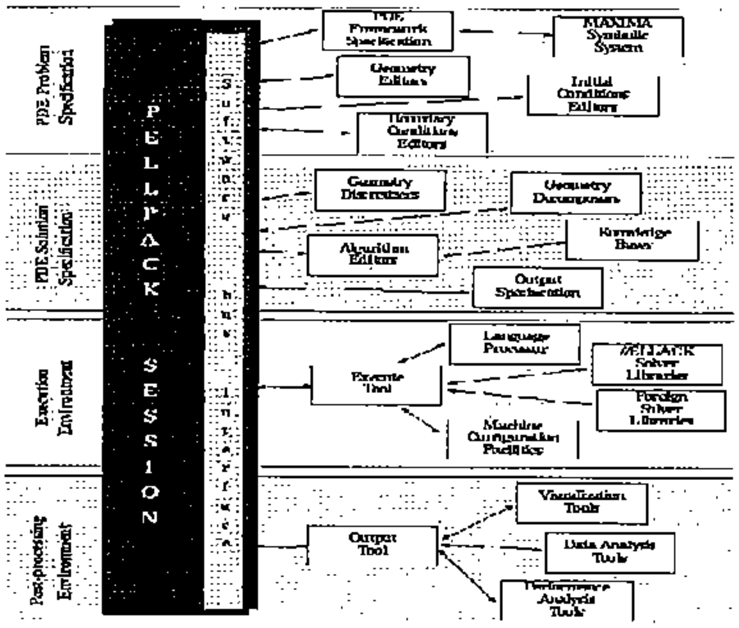


Figure 2. The structure of the PELLPACK PSE. Note that the PDE solvers occupy only two of the 19 boxes here, those at the right side of the execution environment.

II.3 The Alternative Chosen

Since all the "normal" alternatives have some substantial disadvantages, we have chosen a more innovative (and thus more risky) alternative. Its principal advantages are flexibility of GasTurbnLab and semi-automatic parallelism. Note that we do not address the issues of "incomplete physics" in ALE3D or KIVA. We simply do not have the resources to address all such issues, we intend to only enhance our solvers when a particular study needs it. This alternative is presented in the next section.

III GasTurbnLab DESIGN OVERVIEW

The first GasTurbnLab implementation will use three innovative principles:

* Composite (or multi-disciplinary) PDE problems are solved using interface relaxation.
* Domain decomposition is implemented using geometric objects which provide direct access to all user relevant data about the object.
* Agent based computing is used to provide parallelism and asynchronous control of the computation.

III.1 The Geometry Modularity and the User Interface

A consequence of these choices is that the GasTurbnLab GUI is based on geometry modularity. Since GasTurbnLab assumes a given engine (with known geometry, perhaps perturbed in small ways), both the software and user interface are organized by geometry relationships (in space and time). The geometry does have a root node (the entire engine as seen in Figure 3), but there can be multiple ways to subdivide the geometry of a given object. This hierarchy of geometrical objects allows for the wide range of scales that exist in simulations of an engine. Any particular simulation consists of a set of geometric objects which partition the engine, and GasTurbnLab has its PDE solvers collaborate to find a solution for the overall composite PDE problem.

When one starts using GasTurbnLab, one sees the entire engine as in Figure 3. Initially, we will use the 1980's vintage Allison engine XXX. At this level and every lower level, one can view the information (described in Section 4) associated with an object.

III.2 The Network of PDE Solvers

The geometric partition of an engine (or a piece of it) defines a network of PDE problems. On each domain there is a PDE (or a set of them as in the Navier-Stokes equations) that models the physics on that domain. Each domain has some neighbors and, perhaps, some fixed boundaries. If we represent each domain by a box and each neighborhood connection by an arrow linking two domains, we get a network of PDE problems. These represent a composite PDE as the PDEs are normally not the same on each domain. Thus the entire engine (as seen in Figure 3) is represented by the network shown in Figure 4. There are ten connections representing either the continuity of gas flow properties (e.g., temperature, velocity, mass) between the gases in the domains or the mechanical coupling of the two turbines as they drive the fan and compressor.

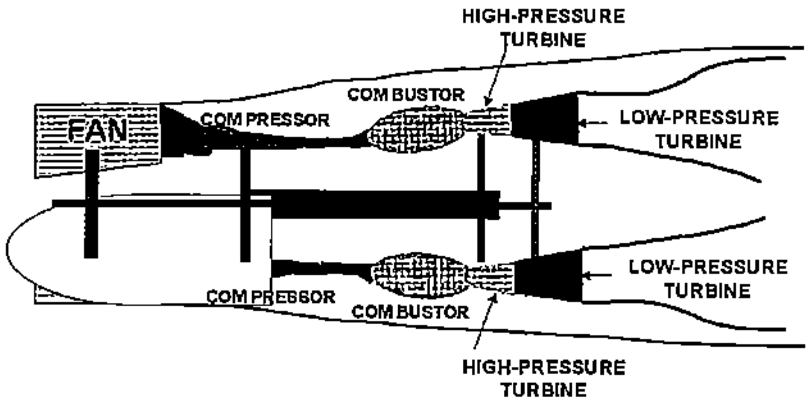


Figure 3. Cross section of the geometry of an engine in GasTurbnLab. Initially only the gas flow components of the engine are considered, i.e., the fan, compressor, combustor, two turbines and the gas mixing zones connecting them. Note that the two turbines are connected mechanically to the fan and compressor.

III.3 Interface Relaxation

Interface relaxation is a mathematical method for solving composite PDE problems (i.e., several different but connected PDEs on several domains) assuming that one can solve **exactly** any single PDE problem. The composite PDE problem is as follows:

> *A set of different PDEs on different but neighboring domains along with interface conditions between neighbors and boundary conditions elsewhere.*

The GasTurbnLab situation seen in Figure 3 is exactly this one. There are nine domains (as named in Figure 4) with gas flow interface conditions between neighboring pairs, plus two mechanical coupling interface conditions. There are also boundary conditions along the inside and outside surfaces of the gas flow area and at the entry and exit of the gas flow.

The iteration of interface relaxation is quite simple:

Interface Relaxation Iteration

Step 1: Guess at solution values, derivatives, etc., on all the interfaces.
Step 2: Solve each PDE exactly with boundary conditions selected from the guesses. There are more interface values available than can be used in solving the PDEs.
Step 3: Compare the solution values across each interface and improve them (using a *relaxation formula*) so as to better satisfy all the interface conditions.
Iterate: Steps 2 and 3 until convergence.

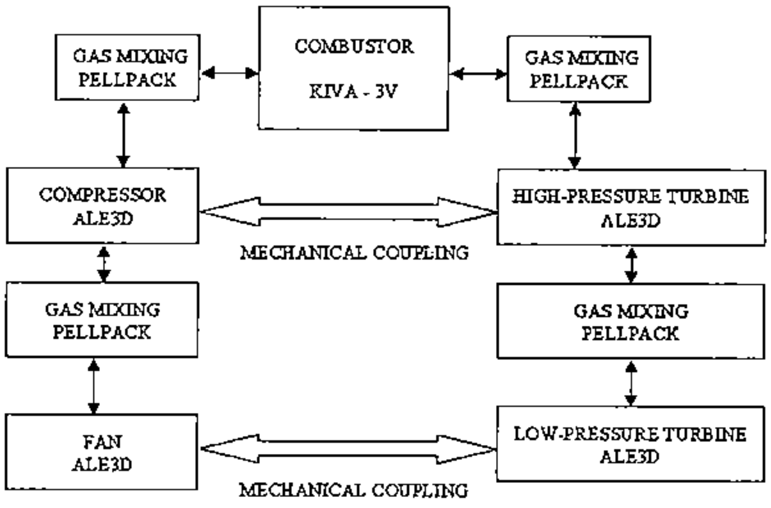


Figure 4. Representation of the network of PDE solvers for the whole engine. The nine PDE problems are individually solvable by the PDE solvers named in the associated box. The ten connections represent either (i) the continuity of gas flow properties between the domains, or (ii) the mechanical coupling of the two turbines to the fan and compressor.

Example: For concreteness, a very simple example is given. We assume (see Figure 5) two domains with one interface and two PDE problems

$$L_1 u_1(x,y,t) = f_1(x,y,z) \quad \text{on} \quad \text{domain} \ \ D_1$$

$$L_2 u_2(x,y,t) = f_2(x,y,z) \quad \text{on} \quad \text{domain} \ \ D_2$$

Along the interface I the conditions are

$$u_1(x,y,t) = u_2(x,y,t) \quad \text{continuity of values}$$
$$du_1/dx = du_2/dx \quad \text{continuity of slopes}$$

and along the boundaries the solutions are known functions. Recall that one can always solve one of these PDEs together with the boundary conditions and **only one** of the two interface conditions. The goal is to find that pair u_1 and u_2 simultaneously satisfies both interface conditions. It is known that there is only one pair of solutions that satisfies both interface conditions.

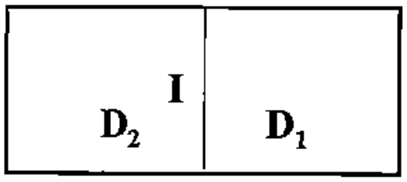


Figure 5. Domains for the example. Boundary conditions are given on the outside boundary (heavy lines) and interface conditions are given on the interface I (thin line).

Suppose now that one has solved the composite PDE problem at time t_0 and one wants a solution at time $t_1 = t_0 + \Delta t$. An interface relaxation method is as follows:

Step 1: Use the values $u_1(x,y,t_0)$, $u_2(x,y,t_0)$, etc., as guesses for the values at time t_1.
Step 2: Solve the PDE problems $L_1u_1 = f_1$, $L_2u_2 = f_2$ with continuity of values (Dirichlet conditions) along I.
Step 3: Compute new interface values with the relaxation formula

$$u_{1,new} = u_{2,new} = (u_1 + u_2)/2 + \alpha(du_1/dx - du_2/dx)$$

where α is an iteration parameter.

Iterate steps 2 and 3 until convergence. Then proceed to time $t_2 = t_1 + \Delta t$, etc.

For more discussion of this method, see [Rice 98], [RTV 99] and the references therein. The state of knowledge about interface relaxation is as follows. There are about 8 or 10 different types of relaxation formulas, most with some parameters. Numerous experiments show that interface relaxation works well for a wide variety of composite PDE problems, but sometimes some relaxation formulas fail. One initial goal of GasTurbnLab is to find a relaxation formula that works very well for the gas turbine simulation PDEs. The mathematical analysis of interface relaxation is very difficult and it is unlikely that much will be proved soon about problems as complex as those in GasTurbnLab.

Finally, we note that in practice one cannot solve the separate PDEs exactly, one must use numerical methods on each of them. However, the numerical methods used can be completely independent of one another.

III.4 The SciAgents System Approach

Experimental systems using interface relaxation has been studied at Purdue for almost a decade. These have evolved into the agent based SciAgents system [DraHo 97], [DHRR 99], more information is available at the web site

`http://www.cecs.missouri.edu/~joshi/sciag/.`

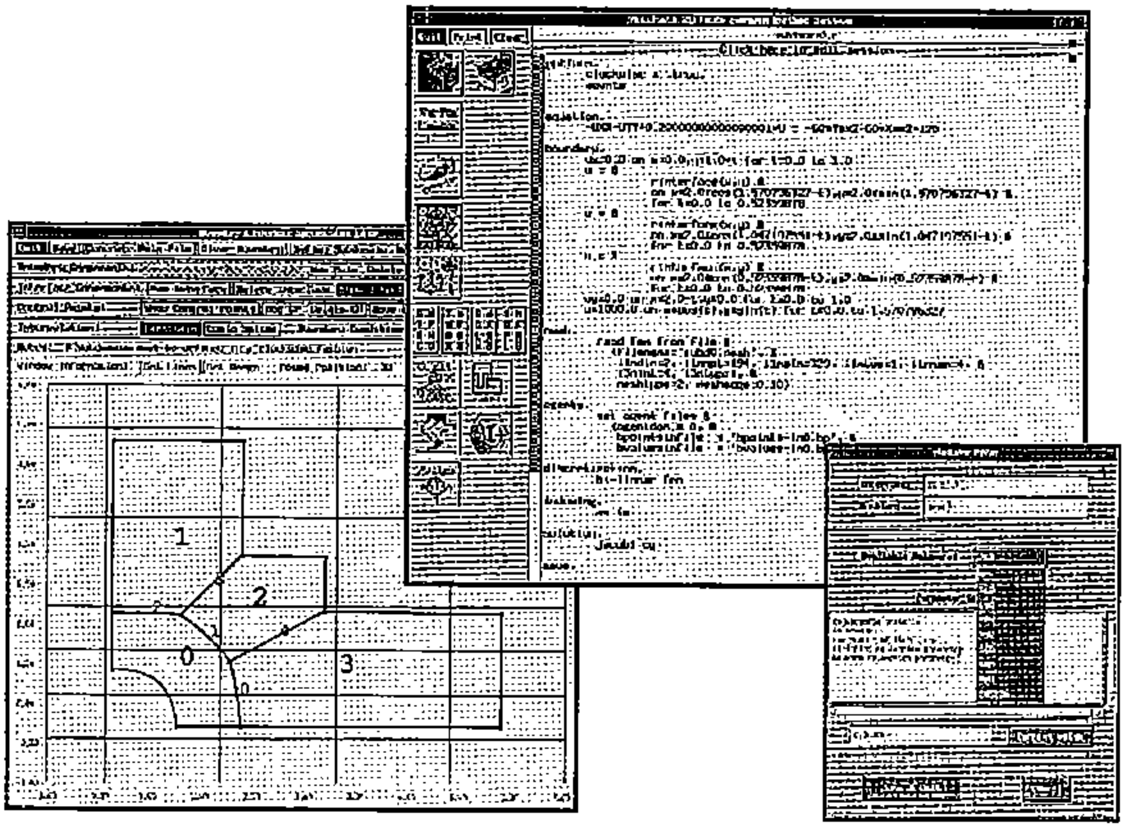
A natural user interface for SciAgents has just been completed [TRV 99], Figure 6 shows sample windows for SciAgents.

The agent based approach uses two types of agents, *solver agents* for the PDEs and *mediator agents* for the relaxation formulas. The GasTurbnLab problem seen in Figures 3 and 4 is represented by the networks of agents seen in Figure 7. This network has 19 agents. Currently SciAgents uses PELLPACK to solve the single PDE problems, i.e., the solver agents are PELLPACK agents. It is relatively simple to transform a PDE solving package into an agent. Making the 2 million line PELLPACK system into an agent required about 1000 lines of new or modified code. SciAgents uses the KQML language and protocol [FFMM 94] for agent interaction; our original selection of an implementation of KQML has proved less robust than hoped and we are investigating other implementations.

The agent based approach is naturally parallel, SciAgents is normally run on a network of workstations with one solver agent per workstation and all the mediator agents on another workstation. Our experience so far is that the computations are very coarse grained; solving a PDE takes 100-1000 times as long as applying the relaxation formulas, even including the network overhead as part of the mediator agent time. Of course, we have not yet run SciAgents with really large numbers of agents and we have not carefully measured the parallel efficiency of SciAgents.

Not only is SciAgents naturally parallel, it is also naturally asynchronous. A simple "have the interface values changed enough" test is used to stop the individual solver agents and the whole computation ends when all the agents have stopped. We regularly see a substantial variation in the number of PDE solutions computed by the individual solver agents.

The SciAgents approach opens up the intriguing possibility that we can make large legacy codes run in parallel without parallelizing them. To be specific, consider the combustor in Figure 3 and that we have made KIVA into an agent capable of simulating a turbine combustor. We can slice the combustor into, say, 10 pieces and assign a KIVA agent to each of them. These 10 agents face a PDE problem almost exactly like the single KIVA agent handling the entire combustor. Thus we can hope to speed up the combustor simulation by a factor of 10 using this simple approach. Of course, we must not let the granularity of the computations become too fine, so as to avoid large slow downs due to communication and interface relaxation costs. The exploration of this possibility is one of the primary goals of the initial use of GasTurbnLab.



```
[illegible]
equation.
    [illegible]
boundary.
    [illegible]
mesh.
    read fem from file [illegible]
agents.
    set agent files [illegible]
discretization.
    bi-linear fem
indexing.
    [illegible]
solution.
    [illegible]
save.
```

Figure 6. Sample windows of the SciAgents user interface. (a) Display of the domains, (b) PELLPACK display to define a single PDE and the numerical method to solve it, (c) Display to select a relaxation formula.

Finally, we note that the SciAgents approach makes it feasible to use a well-known, but rarely used, load balancing technique. If one has 10 tasks assigned individually to 10 processors, then the execution time is the slowest of the 10 tasks. Thus for parallel computing, one is led to divide a large computation into exactly equal subtasks. This is often very hard to do. The alternative is, say for 10 processors, to divide the computation into, say, 50 roughly equal tasks, estimate their individual execution times and then allocate them to the 10 processors so as to balance the load. This allocation process is just bin packing where there are very fast and efficient algorithms. In GasTurbnLab, one is executing the simulation (or applying interface relaxation) for many steps. One can use execution time in formation from the initial steps to redistribute the agents among the processors in order to balance the load.

IV IMPLEMENTATION NOTES

This final section provides additional notes on the GasTurbnLab design.

IV.1 Geometric Objects and Database

Each domain is a geometric object containing the following information:

ID:	Name of the domain, date created, location (both with respect to the coordinate system and with respect to neighbors), type (e.g., combustor, gas flow, blade, atomization jet).
Geometry:	A mathematically exact representation of the geometry.
Partition:	A partition of an object is a set of sub-domains intersecting only at interfaces whose union is the object. Partial partitions are allowed using the complement (i.e., whatever is left over from the partial partition) which is almost empty of information. The sub-domains are geometric objects.
Models:	There are several types:

1. **Mesh**. A partition of the geometry into a discrete mesh or grid. The mesh generator and its parameters are recorded also. This may be an actual mesh or a pointer to a (simple) mesh generator.

2. **Mathematical Models**. These are the equations that represent the physics on the domain. These are usually PDEs.

3. **Numerical Methods**. A synopsis of the numerical method used to obtain the computed solution. These are usually the names of solvers and their parameters.

4. **Solution**. The data of the numerical solution, plus pointers to algorithms that provide values, derivatives, etc., at arbitrary points in the domain. Physically measured data can also be here.

5. **Performance Data**. Information about the accuracy (e.g., residuals, errors), computation resources (e.g., execution times, memory, machine configuration, software systems).

The data in these information categories will be augmented in various ways as GasTurbnLab develops.

These objects are stored in the GasTurbnLab database. Since these objects can be very large and numerous, space will be saved by using pointers (types and names) instead of replication when appropriate, and by trading-off regeneration time for storage space. Incomplete data is allowed, even necessary at times. For example, data from experiments do not have a mathematical model. Care is to be taken to create a new object each time one component is changed (e.g., the grid is modified, a term is added to the PDE, a new iteration parameter is used). If objects become obsolete or flaws are found, they can be deleted from the database.

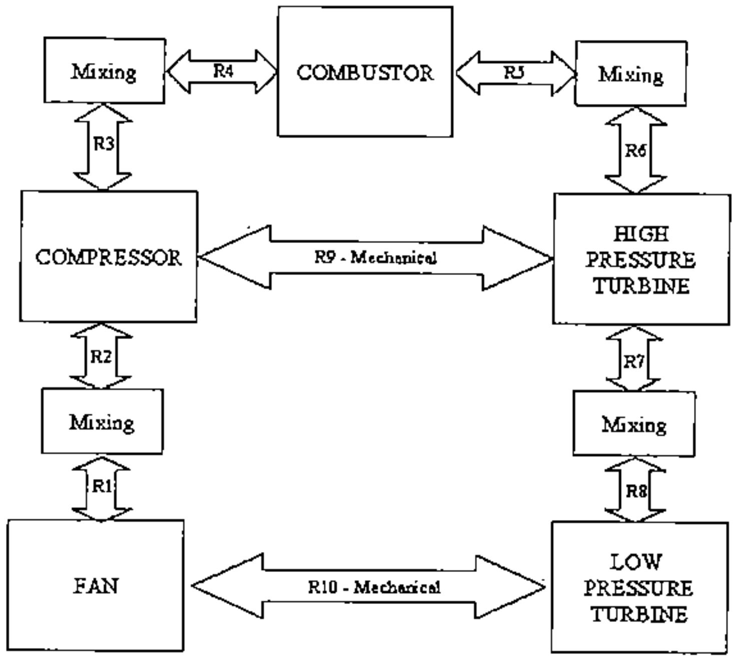


Figure 7. The network of agents for the GasTurbnLab model of Figure 4. The boxes represent the nine solver agents and the arrows represent the ten mediator agents.

IV.2 PDE Solving Agents

GasTurbnLab will have three types of solver agents, the existing PELLPACK agent and two new agents to be created out of KIVA and ALE/ALE3D.

One source of overhead in SciAgents is start-up operations of a solver. PDE solvers are normally written on the assumption that they solve a new PDE problem each time they are executed. In SciAgents, a PDE can be "resolved" several, even many, times with minor, even trivial, changes in the computation. In doing this the solver might regenerate much information, a "worst case" might be for a solver using a direct method which refactors the same numerical model matrix for each iteration. For the solver agents in GasTurbnLab, we will investigate the restart overhead and introduce a restart switch to avoid large recomputations where feasible. On the other hand, we estimate that the start-up overhead is small (a few percent) for most cases, so great effort is not going to be spent on this task. In a similar direction, we will make pre- and post-processing steps optional when convenient as to increase the efficiency of repetitive use of solvers. Modifying a large complex code is always a delicate and time consuming task, but we expect that to make a solver restartable is much simpler than making it parallel.

The geometric object data provides the possibility to copy rather than recompute certain solver information (e.g., a mesh or a partition of a large domain). We will investigate whether it is in fact worthwhile to retrieve such information from the database rather than to recompute it.

IV.3 Reuse of Software Components and Subsystems

The most critical resource in this project is the time of the research staff. Thus we must always be alert to the possibilities for reusing software and reducing the number of software components. For example, we will attempt to use one mesh generator (True Grid) for all three solver agents even if another mesh generator might be marginally (or even somewhat) better in some cases. This approach will be used for various components of pre/post-processing (e.g., visualization, symbolic processing, mesh generation) and the system infrastructure (e.g., the agent system, database system, interpolation and data transformation, object manipulation). We will systematically peruse the DOE ASCI software for programs and systems like

1. AMR++ (Adaptive Mesh Refinement class library),
2. A++/P++ (C++ class libraries for arrays),
3. CDMlib (Common Data Model library),
4. Overture, (C++ class libraries for overlapping grids),
5. PDT (Program Database Toolkit),
6. PDVF (Parallel Distributed Visualization Framework),
7. POOMA (Parallel Object-Oriented Methods and Applications),
8. POPTEX (High Performance Interactive Visualization Tool),
9. SILOON (Scripting Interface Languages for Object-Oriented Numerics),
10. SMARTS (Shared Memory Asynchronous Runtime System),

that we might incorporate easily into GasTurbnLab. See [DOE 99] for URLs.

V REFERENCES

[DOE 99]

`AMR++`	= `http://www.c3.lanl.gov/~dquinlan/AMR++.html`
`PAWS`	= `http://www.acl.lanl.gov/PAWS/`
`Pooma`	= `http://www.acl.lanl.gov/pooma/doc/userguide/poomaUserGuide-1.html`
`Overture`	= `http://www.c3.lanl.gov/~dlb/napc/documentation.html`

[DHRR 99] T.T. Drashansky, E.N. Houstis, N. Ramakrishnan, and J.R. Rice. Networked agents for scientific computing, *Comm. Assoc. Comp. Mach.* (1999).

[DraHo 97] T.T. Drashnansky, E.N. Houstis, A. Joshi, J.R. Rice, and S. Weerawarana. Collaborating problem solving agents for multi-physics problems, *15th IMACS World Congress*, Wissenschaft & Technik Verlag, 4 (1997), 541–546.

[FFMM 94] R. Fritzson, T. Finn, D. McKay, and R. McEntire. KQML-A language and protocol for knowledge and information exchange, *Proc. 13th Inter. Distributed Intelligence Workshop*, Springer-Verlag, New York (1994).

[HoRice 98] E.N. Houstis, J.R. Rice, S. Weerawarana, A.C. Catlin, P. Papachiou, K.Y. Wang, and M. Gaitatzes. PELLPACK: A problem solving environment for PDE based applications on multicomputer platforms, *ACM Trans. Math. Software*, **24** (1998), 30–73.

[HoRice 98a] E.N. Houstis, J.R. Rice, N. Ramakrishnan, T. Drashansky, S. Weerawarana, A. Joshi, and C.E. Houstis. Multidisciplinary problem solving environments for computational science. In *Advances in Computers*, vol. **46**, (M.V. Zelkowitz, ed.) (1998), 401–438.

[Joshi 97] A. Joshi, T. Drashansky, J.R. Rice, S. Weerawarana, and E.N. Houstis. Multi-agent simulation of complex heterogeneous models in scientific computing, *Math Computers Simulation*, **44** (1997), 43–59.

[MuRice 95] M. Mu and J.R. Rice. Modeling with collaborating PDE solvers – Theory and practice, *Comp. Syst. Engr.*, **6** (1995), 87–95.

[Rice 98] J.R. Rice. An agent based architecture for solving partial differential equations, *SIAM News*, **31**, No. 6 (1998), 1,6–7.

[RVY 97] J.R. Rice, E.A. Vavalis, and D. Yang. Analysis of a non-overlapping domain decomposition method for elliptic partial differential equations, *J. Comp. Appl. Math.*, **87** (1997), 11–19.

[TRV 99] J.R. Rice, P. Tsompanopoulou, and E. Vavalis. Interface relaxation methods for elliptic differential equations, *J. Appl. Numer. Meth.* (1999), to appear.

[TRV 99] P. Tsompanopoulou, J.R. Rice, and E. Vavalis. The SciAgents user interface, CSD-TR 99-XXX, Dept. of Computer Sciences, Purdue University.