Networked Agents for Scientific Computing*

Tzvetan Drashansky
Juno Online Services, L.P.
120 West 45th Street
New York, NY 10036
email: ttd@staff.juno.com,ttd@deshaw.com

Elias N. Houstis
Department of Computer Sciences, Purdue University
West Lafayette, IN 47907-1398
email: enh@cs.purdue.edu

Naren Ramakrishnan
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
email: naren@cs.vt.edu

John R. Rice
Department of Computer Sciences, Purdue University
West Lafayette, IN 47907-1398
email: jrr@cs.purdue.edu

1 Overview

The thesis of this article is that agent based computing provides important advantages for scientific computing. We present this thesis in the context of a particular application, the simulation of gas turbine engines. This application is typical in that it involves an enormously complex device of great economic importance, one whose design is continually evolving to achieve higher value and to fit new uses.

Ideally, a designer would change some aspect of the engine and then run a simulation to see how the performance, cost, durability, etc., change. Such a simple approach will be infeasible for the foreseeable future because the complete simulation of an engine design requires days, weeks or even years on petaflops class computing systems. Thus the design process and simulation software must be configurable so that a simulation can focus on particular aspects of the engine. For example, in designing a new turbine blade (they do not all have the same shape) one might (a) model the blade itself very accurately, (b) model the air flow field and structure near the blade with moderate accuracy, (c) model the air flow fields and structures further away roughly, and

*This research was supported in part by the National Science Foundation (CDA–9123502, CCR–9311486) and the Defense Advanced Research Projects Agency through the Army Research Office (DAAH04-94-G-0010).

(d) model the remainder of the engine by fixed boundary conditions surrounding the focal area of this particular simulation. Step (d), of course, removes over 99% of the parts and phenomena from this particular simulation, making it feasible to explore many design parameter effects quickly. As a new engine design evolves and matures, the focus of such simulations is enlarged, first bringing several new parts together, then dozens or hundreds of parts, then complete subassemblies, and, finally, the entire engine. Concurrent with this bottom up approach there will be top down simulation effort where 'everything' is modeled, though roughly (many parts or assemblies are merged into single simulation components or simply ignored) to obtain initial, rough estimates of overall behaver. Further, as difficulties or shortcomings appear, various combinations of these two approaches are used. Moreover, this form of prototyping requires knowledge and computational models from multiple disciplines.

Sidebar Begin: Turbines

Cutaway view of gas turbine engine for an airplane (courtesy Sanford Fleeter). Some of the performance and complexity characteristics of this device are indicated. They typically have tens of thousands of parts, many of which experience extreme operating conditions. Important physical phenomena take place at space scales from tens of microns (combustion, turbulence, material failure) to meters (gas flow, rotating structures) and at time scales from microseconds to months.

The analysis of an engine involves the domains of *thermodynamics* (gives the heat flows throughout the engines), *reactive fluid dynamics* (gives the behavior of the gases in the combustor), *mechanics* (gives the kinematic and dynamic behaviors of pistons, links, cranks etc.), *structures* (gives the stresses and strains on the parts) and *geometry* (gives the shape of the components and the structural constraints). The design of the engine requires that these different domain–specific analyses interact in order to find the final solution. While these different domains might share common parameters and interfaces, each of them is governed by its own constraints and limitations.

- There are thousands of well defined modules for modeling various parts and behaviors or for supporting the simulation process (e.g., visualization, data management, module interfacing).
- For most design aspects there are multiple software modules to choose from. These embody different numerical methods (iterative or direct solvers), numerical models (standard finite differences, collocation with cubic elements, Galerkin with linear elements, rectangular grids, triangular meshes), and physical models (cylindrical symmetry, steady state, rigid body mechanics, full 3D time dependent physics).

Sidebar End: Turbines

1.1 Distributed Problem Solving

The first step to modeling is to replace the original multiphysics problem by a set of smaller simulation problems on simple geometries which need to be solved simultaneously while satisfying a set of interface conditions. These simpler problems may be chosen to reflect the underlying structure/geometry/physics of the system to be simulated, or artificially created by scientific computing techniques such as domain decomposition. For physical systems and devices, these subproblems are usually modeled by partial differential equations (PDEs). The next step is to create a network of collaborating *solver agents* where each such agent deals with one of the subproblems defined earlier. The original multiphysics problem is solved when one has all the equations satisfied on the individual components and these solutions 'match properly' on the interfaces between the components. Matching is the responsibility of *mediator agents* which facilitate the collaboration between solver agent pairs. The term 'match properly' is defined by the physics if the interface is where the physics changes. For heat flow, for example, this means that temperature is the same on both sides of the interface and that the amount of heat flowing into one component is the same as the amount flowing out of the other. If the interface is artificially introduced to make the computations simpler, or to exploit parallel computation, then 'match properly' is defined mathematically and means that the solutions join smoothly (have continuous values and derivatives). Distinguishing solver and mediator agents enables us to handle complex mathematical models naturally and directly. The agents can be organized into hierarchical structures that reflect the physical structure of, say, the jet engine. Further, this hierarchy allows alternative models and solvers to facilitate the many locally focused simulations to be made.

Other approaches to distributed problem solving have been proposed for scientific computing, particularly ones that focus on harnessing distributed object oriented technologies. The Globus system [5] provides basic software infrastructure for computations that integrate geographically disparate resources. The Legion project [8] provides an object–based meta–system to enable third party development of scientific applications and infrastructure components. The Infospheres project [2] at Caltech provides a distributed programming layer using the World Wide Web, Java and Internet. These operate at a lower level of abstraction and hence, the ideas illustrated here can utilize them as back-ends to facilitate interaction and coordination among the agents. This methodology is a natural component of metacomputing, see [9] and related papers in the November issue of the Comm. ACM.

2 Resource Selection

For a complex device, the solver and mediator agents form a large pool of computational objects spread across the network. The past few decades has seen a huge amount of sophisticated code being developed to solve specific, homogeneous problems. Clearly, expecting a solver/mediator agent to be aware of all the potentially useful software on the Net is not realistic. Nor is a single user likely to know all the hardware choices available to solve the problem. To circumvent this resource selection bottleneck, we introduce *recommender agents* which serve as 'advisory' engines. The purpose of a recommender agent is to accept a query from a solver (or mediator) agent about a problem, determine a suitable algorithm that applies to that problem and finally, direct (recommend) it to an appropriate location on the Net where software implementing the algorithm can be obtained and executed. This is similar to the idea of recommender systems that harness distributed information resources on the Internet [12]. Such agents are extensively used in commercial search engines and web–based data warehouses.

The organization of software on the Net and tracking their availability is facilitated by cross–indices of mathematical software such as GAMS [1]. GAMS is a virtual mathematical software repository that helps users to identify and locate the right pieces of software for their problem. However, the user has to select the specific routine most appropriate for the given problem, download the software along with its installation and use instructions, install the software, compile (and possibly port) it, and then learn how to invoke it appropriately. The recommender agents serve to automate many aspects of of this software/resource selection process.

Even after the choice of software/algorithm is made, the specialized facilities/platform requirements of the software might not be available to the user locally. In such cases, the user must access remote computational servers that can act as (or serve) the agents performing the computations. In other words, the solvers and mediators needed for the simulation should be accessible over the Net and recommender agents should be able to dynamically adapt and determine the most suitable location(s) to perform computations.

Sidebar Begin: Tools Available for Scientific Computing

GAMS [1] is a virtual mathematical software repository system with access to thousands of software modules. The gas turbine design process would use a version of GAMS specialized and enlarged for this application. It promotes easy access and also encourages software reuse in the current distributed computing environments. GAMS's main contribution to mathematical software, however, lies in the tree structured taxonomy of mathematical and software problems used to classify software modules. This taxonomy extends to seven levels and provides a convenient interface to home in on appropriate modules. For example, the problem class I refers to modules catering to differential and integral equations, I2 indexes to modules about PDEs, I2b caters to elliptic boundary value problems, I2b1 refers to the linear kind of these problems and so on. This taxonomy indexes about $10,000$ problem solving modules from over 100 software packages maintained at four software repositories on the Internet. Much of this software is in FORTRAN format for problems from linear algebra, differential equations, number theory, optimization, statistics, interpolation, approximation, symbolic computation, geometry and most other areas in computational science.

Sidebar End: Tools Available for Scientific Computing

In our implementation, the PYTHIA recommender agents help select the algorithm and its executable implementation. Each recommender agent can provide recommendations for a certain class of problems and can also collaborate with other agents to collectively arrive at a recommendation. For example, with a certain input, an agent might provide the recommendation: '*Use the 5–point star algorithm with a* 200×200 *grid on an NCube/2 using* 16 *processors: Confidence –* 0.85. *Software available at* `http://math.nist.gov/cgi-bin/gams-serve/list-module-components/ELLPACK/1-14-46/13058.html`'.

The PYTHIA system provides the recommender agents needed for multidisciplinary simulation. A PYTHIA agent [13] gathers performance information about solvers on standardized test problems and uses this data (plus features of existing problems) to determine good algorithms to solve a newly presented problem. The efficacy of a single agent is thus dependent on the methods and the problem sets referenced in the performance information collected by it and its ability to determine the features of the new problem. PYTHIA's methodology recognizes that there are many different kinds of problems but that most recommender agents are able to advise for only a limited subset of those in an application domain. When an agent's expertise is exhausted, it solicits recommendations from all the other agents and chooses the one that has the highest degree of 'reasonableness' [7, 11]. Collaboration between the agents is facilitated by a language called PYTHIA–Talk that is based on the KQML model of agent communication [4]. PYTHIA–Talk describes the meaning (content) of the KQML messages while the underlying communication infrastructure is provided by the TCP transport protocol. Our implementation also provides facilities for agents to dynamically enter and leave the system.

This prototype implementation relates to a variety of PDEs known as elliptic PDEs. At the outset, there is a facility to provide feature information about a PDE problem. In particular, there are forms that enable the user to provide details about the operator, function, domain geometry and boundary conditions — basic parts of a PDE. Once these details are provided, a PYTHIA agent starts its task by first classifying the given problem into several categories of problems. It uses this classification to determine whether its expertise could be used to provide recommendations or if other agents need to be contacted. If needed, the PDE is transferred to another PYTHIA agent appropriate for its class of problems, which in turn predicts an optimal strategy to finally report back to the original agent and the user. This process is repeated, if necessary, to arrive at a good solution. Having performed software selection, PYTHIA now interfaces with the GAMS system to direct the user to an implementation of the recommended algorithm.

The interface between PYTHIA and the GAMS repository forms the basis of a collaborative software selection facility [10]. This interface uses the GAMS taxonomy of mathematical software to categorize the PYTHIA agents and their capabilities. Moreover, the PYTHIA agents are themselves based on extensive performance evaluation of GAMS-indexed software, so the combination is very pertinent. The interface serves a dual purpose: to direct PYTHIA users to appropriate GAMS locations for their software needs, and also to provide advisory support to GAMS users.

In our implementation, the connection between the PYTHIA system and Gams is achieved by a proxy software program that is configured for this purpose. Requests sent to the GAMS server about methods for some categories of problems (e.g., elliptic PDEs of the I2b1a1 category) are transferred to the appropriate PYTHIA agent by the proxy. Thus the proxy serves to enforce acceptability criteria on the user's input. At this point, the PYTHIA agent in turn requests input from the user about the PDE problem characteristics and recommends a method to solve the given PDE. In addition to displaying this information, it also provides a hyperlink back to the GAMS system to download the appropriate software modules. Our facility also provides a convenient scheme to switch back and forth between the GAMS and PYTHIA systems.

3 *SciAgents*

While PYTHIA supplies the recommender agents that interface with GAMS, the solver and mediator agents are provided by the *SciAgents* system [3]. Each solver agent is considered a 'black box' by the other agents and it interacts with them using an interagent language. This allows all computational decisions for solving one individual subproblem to be taken independently from the decisions in any other subproblem – a major improvement over the traditional approaches to multidisciplinary simulations. Each mediator agent is responsible for adjustments at an interface between two neighboring subproblems. The interface between two subproblems might be complex, so more than one mediator might be assigned, each operating on a separate piece of the interface. Thus the mediators control the data exchange between the solvers working on neighboring subproblems by applying mediating formulas and algorithms to the data from the solvers.

SciAgents provides a robust mechanism for cooperation among the computing agents. The agents perform only local computations and communicate only with neighboring agents. They *cooperate* in solving a global, complex problem, and none of them exercises centralized control over the computations. The global solution 'emerges' in a well-defined mathematical way from the local computations as a result of intelligent decision making done locally and independently by the mediator agents. The agents may change their goals dynamically according to the local status of the solution process – switching between observing results and computing new data.

SciAgents operates in conjunction with the Parallel ELLPACK Problem Solving Environment (PSE) [6]. Parallel ELLPACK is a scientific software server that provides a natural interface for simulations based on partial differential equations. In addition, Parallel ELLPACK tracks extended problem–solving tasks and allows users to review them easily by exploiting modern technologies such as interactive color graphics, computation steering, parallel processors and networks of specialized services.

Figure 1: *SciAgents* applied to a problem with four subdomains with different PDEs. (left) A snapshot of the display during the subproblem definition process. Parts of three Parallel ELLPACK domain tools are shown containing subdomain geometries and finite element meshes. Each subdomain is discretized independently from the others. (right) A combined picture of all subdomain solutions. The global solution corresponds to the physical intuition about the behavior of the modeled real-world system.

We illustrate the performance of *SciAgents* by applying it to the solution of a composite problem that involves heat distributions in the walls of an engine part and in the surrounding isolating and cooling structures. Once the subdomains (and hence, the solvers), their properties, and the mediators are defined, *SciAgents* builds a network of solvers and mediators to solve the problem. A global controller provides the 'navigation' necessary to steer the computations. When a mediator observes convergence (the change of the boundary conditions for the next iteration is smaller than the tolerance), it reports this to the global controller, and after all mediators report convergence, the global controller issues a message to all agents to stop. Figure 1 describes a sample input problem definition to *SciAgents* and the final solution obtained after convergence. It can be observed that all contour lines match when crossing from one subdomain to another, there are even a few which go through three subdomains and one going through all four subdomains. This demonstrates the efficacy of our collaborative approach to problem solving.

The multi agent architecture presented here, enables us to combine existing full fledged problem solving environments and software libraries to realize large–scale multidisciplinary simulation facilities. We are optimistic that systems like *SciAgents* and PYTHIA will provide the core functionality necessary to achieve these goals. We refer to these new facilities as 'multidisciplinary problem solving environments'.

Our ongoing research focuses on many more aspects of these problems. *SciAgents* is being extended to address more problem domains and the recommender agents of PYTHIA are being used to satisfy more software selection needs. In addition, issues in learning and adaptation in multi–agent systems are being explored to aid in better coordination control between the various types of agents. Together, they promise to address one of the most important challenges in scientific computing.

[1] R.F. Boisvert, S.E. Howe, and D.K. Kahaner. The Guide to Available Mathematical Software Problem Classification System. *Communications in Statistics - Simulation and Computation*, volume 20(4):pages 811–842, 1991. accessible at `http://gams.nist.gov`.

[2] K.M. Chandy. The Scientist's Infosphere. *IEEE Computational Science & Eng.*, 1996.

[3] T.T. Drashansky, A. Joshi, and J.R. Rice. SciAgents - An Agent Based Environment for Distributed, Cooperative Scientific Computing. In *Proceedings. 7th Intl. Conf. Tools with Artificial Intelligence*, pages 452–459. IEEE Computer Society, 1995.

[4] T. Finin, Y. Labrou, and J Mayfield. KQML as an Agent Communication Language. In J. Bradshaw, editor, *Software Agents*. MIT Press, Cambridge, 1997.

[5] I. Foster and C. Kesselman. The Globus Project: A Status Report. In *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, pages 4–18, 1998.

[6] E.N. Houstis, J.R. Rice, S. Weerawarana, A.C. Catlin, P. Papchiou, K-Y. Wang, and M. Gaitatzes. PELLPACK: A Problem Solving Environments for PDE–Based Applications on Multicomputer Platforms. *ACM Transactions on Mathematical Software*, 24(1):pages 30–73, 1998.

[7] A. Joshi. To Learn or Not to Learn ... In G. Weiss and S. Sen, editors, *Adaptation and Learning in Multiagent Systems*, volume 1042 of *Lecture Notes in Artificial Intelligence*, pages 127–139. Springer Verlag, 1996.

[8] M.J. Lewis and A. Grimshaw. The Core Legion Object Model. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society Press, August 1996.

[9] S. Karin and S. Graham, *The High-Performance Computing Continuum*, 41(11):pages 32–35, 1998.

[10] N.Ramakrishnan, E.N. Houstis, A. Joshi, J.R. Rice, and S. Weerawarana. Intelligent Networked Scientific Computing, *Proc. 15th IMACS World Congress*, vol. 4:pages 785–790, Wissenschaft and Technic Verlag, 1997.

[11] N. Ramakrishnan, A. Joshi, E.N. Houstis, and J.R. Rice. Neuro–Fuzzy Approaches to Collaborative Scientific Computing. In *Proceedings ICNN'97*, pages 473–478. IEEE Press, 1997.

[12] P. Resnik and H.R. Varian. Recommender Systems. *Communications of the ACM*, vol. 40(3):pages 56–58, 1997.

[13] S. Weerawarana, E.N. Houstis, J.R. Rice, A. Joshi, and C.E. Houstis. PYTHIA: A Knowledge Based System to Select Scientific Algorithms. *ACM Transactions on Mathematical Software*, volume 22(4):pages 447–468, 1996.