1997

Solving PDEs Using An Agent Based Architecture

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

Report Number:
97-043

SOLVING PDEs USING AN AGENT BASED ARCHITECTURE

John R. Rice

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD-TR #97-043
September 1997

SOLVING PDEs USING AN AGENT BASED ARCHITECTURE

John R. Rice, September 29, 1997

Composite or multi-physics partial differential equation (PDE) problems arise from modeling where multiple physical phenomena are involved. Boiling a pan of water, for example, involves heat conduction in solids, plus both heat conduction and heat convection in water and in air. Figure 1 shows a similar device with the same three physical phenomena. While the individual physical phenomena are modeled by PDEs in the usual way, their solutions must satisfy interface conditions between the domains of the phenomena. These interface conditions also model the physical behavior at the interfaces. In our simple example, the interfaces conditions correspond to the continuity of temperature and heat flux. If the water starts to boil, then the air-water interface becomes much more complicated as mass (steam) transfers from the water into the air carrying heat with it. One of the challenges at the frontier of simulation technology is to solve the composite PDE equations that arise from such models in more complicated devices (e.g., a complete internal combustion or gas turbine engine).

We describe the *interface relaxation method* for solving composite PDEs and then present a (virtual) architecture and software system *SciAgents* [3] that uses this method and architecture. Assume that one can solve exactly any single PDE on any simple domain, or, more realistically, given such a PDE problem one can select an exact solver for it from one's library. Interface relaxation is a method to solve composite PDE problems using a library of "single, simple domain, exact" PDE solvers. It is an iterative method of the classical type based on relaxation as follows:

1. *Guess solution values (and derivatives if needed) on all interfaces.*
2. *Solve all single PDEs exactly and independently with these values as boundary conditions.*
3. *Compare and improve the values on all interfaces using a* **relaxer** *(discussed below).*
4. *Return to Step 2 until satisfactory accuracy is achieved.*

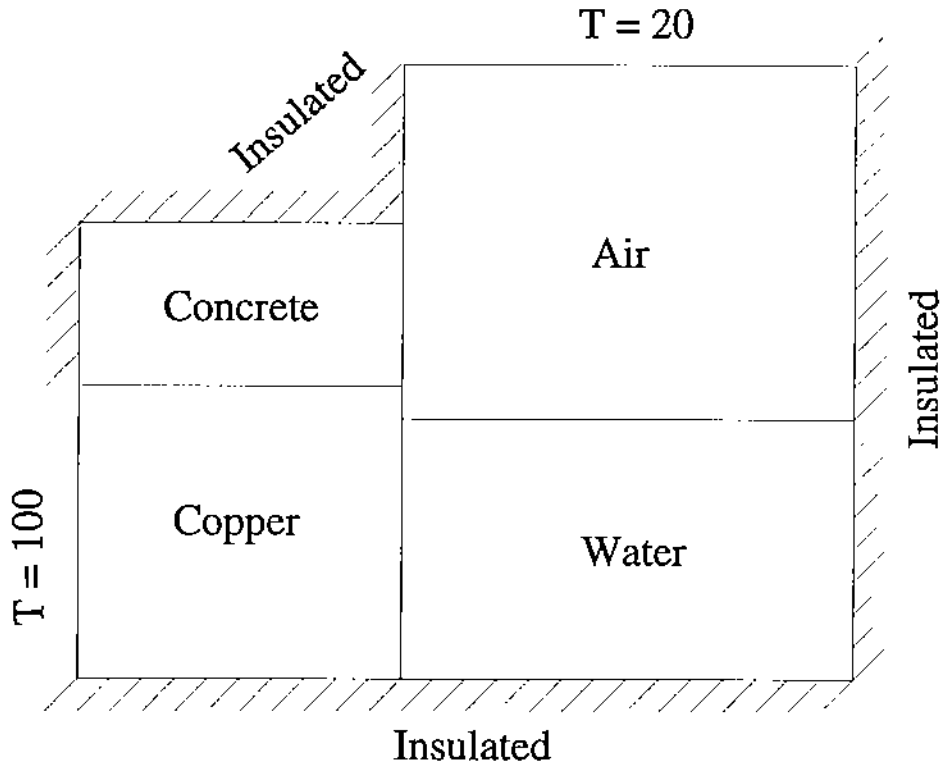

Figure 1: A simple composite PDE problem with three phenomena, four simple domains, and five interfaces. Heat is transmitted across the interfaces according to standard models of heat flux and temperature T continuity.

Thus interface relaxation is in the spirit of Southwell's idea as applied to linear algebraic equations used in the 1930s. The simplest relaxers just take some sort of "average" values on the interfaces and that is a good mental model for a relaxation formula.

The attraction of the SciAgents architecture is threefold. First it allows the accurate coupling of independent models and the reuse of PDE software that handles single phenomenon models. Second it uncouples, somewhat, the parallelism of the computation from that of the machines used. Finally, it is naturally related to a person's view of the geometry and physical models of a composite PDE problem. Interface relaxation is an iteration defined at the continuum (mathematical) level; its convergence properties are a question of mathematical analysis and not of numerical analysis. Let U_N and V_N be the PDE solution values at iteration N on opposite sides of an interface and assume, for simplicity, that the interface conditions to be satisfied are continuity of value ($U_N = V_N$) and normal derivative ($\partial U_N/\partial n = -\partial V_N/\partial n$). Then two simple relaxation formulas for U and V

are

$$V_{N+1} = U_{N+1} = (U_N + V_N)/2 - f * (\partial U_N/\partial n + \partial V_N/\partial n)$$
$$V_{N+1} = U_{N+1} = \omega U_N + (1-\omega)[\alpha(U_N - V_N)^2 + \beta(\tfrac{\partial U_N}{\partial n} + \tfrac{\partial V_N}{\partial n})^2]$$

where f, w, α, and β are relaxation parameters. Early relaxation formulas were proposed in the context of domain decomposition (e.g., [6], [5]), the talk [9] catalogs about 10 relaxation formulas. One of the important open questions is their comparative performance and how to compute good relaxation parameters. Of course, actual implementations of interface relaxation use numerical methods because the exact mathematical PDE solvers do not exist. One should visualize that the numerical solvers used instead produce a substantially higher level of accuracy than that of the relaxation iteration so there is negligible interaction between the accuracy of interface relaxation and that of the numerical solvers. It is like assuming that computer arithmetic is exact even though it is not (and this can sometimes cause problems). Interface relaxation looks like *domain decomposition* but it is not because domain decomposition has a single underlying PDE for all the domains and the interface conditions are mathematical smoothness (continuous value and derivative) or some numerical equivalent.

The convergence analysis of interface relaxation presents formidable mathematical challenges; almost any question one asks is both hard and open. Even for the single PDE case (one global PDE or domain decomposition) only a few analysis efforts have been made starting about 10 years ago by P.L. Lions [6] and A. Quarteroni [8] then more recently by M. Mu [7] and J. Douglas [2]. The analysis for this case is greatly simplified because there already exists a rich convergence theory for solving the single PDE that can be applied. The difference between the two convergence problems is illustrated as follows: let U_{Nh} or U_{hN} be the solution produced at iteration N by discretization h (h is merely a placeholder for interface relaxation since there is no concept of a global discretization variable). The two methods and their convergence questions are then:

Domain Decomposition	*Interface Relaxation*
1. Discretize PDE using h to define U_h	1. Problem is decomposed into subdomains
2. Decompose problem into subdomains	2. Use interface conditions and iteration method to define U_N
3. Use interface conditions and iteration method to define U_{hN}	3. Discretize each PDE using h to define U_{Nh}
4. Iterate 3 on N until convergence	4. Let $h \to 0$ in 3 until convergence
5. Let $h \to 0$ in 1–4 until convergence $\lim_{h\to 0}\ (\lim_{N\to\infty} U_{hN})$ =??	5. Iterate 2–4 on N until convergence $\lim_{N\to\infty}\ (\lim_{h\to 0} U_{Nh})$ =??

For domain decomposition if one can handle $\lim_{N\to\infty}$ then the other limit can be handled by existing theory. This is not so for interface relaxation since there is no convergence theory for multiple (or even one) discretization method applied to composite PDEs.

Given that theoretical analysis is intractable for the moment, one should use experiments to provide guidance and insight for interface relaxation. Numerous experiments have been made in recent years which indicate that interface relaxation converges for a wide variety of problems and relaxers. Sometimes the convergence is very fast, sometimes it is not. There is reason to be hopeful that, as we better understand interface relaxation, it can become a very useful method for solving composite PDEs. Note that a crude form of interface relaxation is already in fairly widespread use. That is just to "trade" current values across interfaces without any relaxation. This makes the most sense in time varying problems but I have not seen anyone try to analyze the effect of the errors involved.

Interface relaxation is naturally suited for distributed high performance computing. The method defines a mathematical network with a single PDE solver at each node (representing a domain) and relaxers connecting the nodes. One distributes the single (and usually different) PDE solvers to high performance machines (since many PDE problems have to be solved) and let the relaxers "control" the computation. One can extend this by making each node a *solver agent* capable of solving its PDE and its goal is to do this whenever new boundary (interface) conditions are given it. One can make each relaxer connection into a *mediator agent* capable of accepting values from solvers, applying relaxation formulas, and returning improved values to its two solvers. Its goal is to apply these formulas and to decide if convergence has taken place (do the values it receives satisfy the interface conditions sufficiently well?). The entire method is placed into an agent based framework by introducing a control agent which sets policies for the other agents, e.g., what are the tolerances for local convergence? should a solver agent resolve its PDE when it gets each new interface value or wait until it gets all new interface values?

This agent based framework creates a virtual architecture for each particular composite PDE problem. It must be mapped onto a real architecture and then other issues come into play, e.g., load balancing or communication latency. If one of the nodes has a dominating PDE solution time, then this local PDE problem may be partitioned using traditional domain decomposition methods. However, an attraction of the agent based approach is that it cleanly separates the algorithmic and software issues from the hardware being used.

SciAgents is an agent based implementation of interface relaxation built on the PDE solving

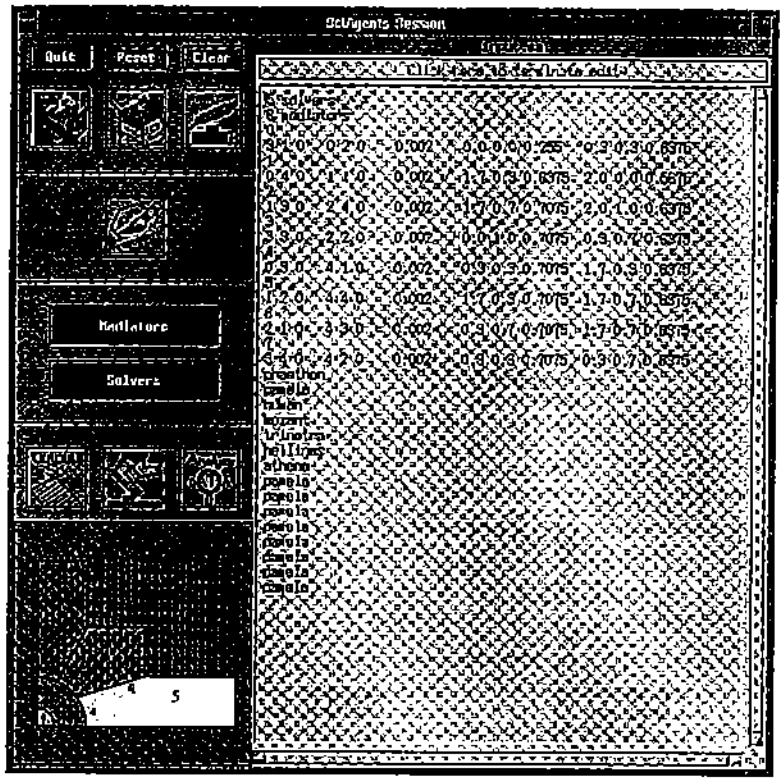

Figure 2: The SciAgents control window.

infrastructure of *Parallel ELLPACK* (PELLPACK) [5]. It has about a dozen PDE solving packages (e.g., CADSOL, ELLPACK, FIDSOL, NSC2KE, PDECOL, VECFEM) integrated into a uniform problem solving environment; some of these, in turn, contain many individual solvers so the overall system contains over 1.5 million lines of code. For any particular PDE, of course, only one solver is used. While PELLPACK has 1-D, 2-D, and 3-D solvers, SciAgents is currently restricted to 2-D problems. For SciAgents a *wrapper* has been put around PELLPACK to make it into an agent. This wrapper involves adding or changing about 1000 lines of code in PELLPACK, an almost negligible amount compared to the total code.

SciAgents has a facility to create the network of solver and mediator agents for a composite PDE problem. It has templates for mediator agents where one can use a default relaxer, select one from a menu, or (for experts) program a new relaxation formula. Thus the creation facility relies heavily on the user interfaces (GUI) of the solver and mediator agents. The PELLPACK solver GUI is elaborate and provides many options for defining PDE problems, selecting among the solvers

and visualizing the solution. The SciAgents mediator GUI is much simpler, allowing one to select one of eight built-in relaxation formulas or to program a new one. To create a SciAgents network one makes a sketch of the composite PDE problem so as to identify (and number) the domains and interfaces. Then the user uses the agent GUI one by one (or by groups) to define the network, provide equations and parameters, etc. An example at the end of the process is shown in Figure 2 where the domain is displayed at the lower left. The window shows the (somewhat cryptic) text of SciAgents where the network data is consolidated. Note at the bottom of this text that machine names are given for each agent, as yet SciAgents does not provide any automatic load balancing or automatic machine assignment facilities. SciAgents is implemented using the KQML system [4] for information exchange which has been used for a number of agent based applications in the AI community. The SciAgents application stressed KQML (the KAPI implementations) in two respects, both of which one would expect to be common in agent based systems in science. First, the size of the information packets quickly exceeded the KQML limits and a "work-around" was implemented early on. Second, the KQML system degrades and then crashes as the number of inter-agent messages grows. Thus the KQML based implementation of SciAgents becomes unreliable at about 8 to 10 domains. We plan to use a more robust KQML implementation or a substitute system in the future. The SciAgents architecture for the composite PDE problem of Figure 1 is shown in Figure 3. The system components are shown along with the problem solving agents created for this application.

By now we have solved perhaps 200 2-D composite PDEs using interface relaxation. Perhaps 150 of these problems are of domain decomposition type, i.e., the same elliptic PDE is used on all domains. In these experiments we vary the PDE, the number of domains (up to 500 have been used), the shapes of domains, relaxation formulas, etc. The other 50 have been truly composite PDE problems, usually linear elliptic PDEs. Again the objective was to explore the range of applicability of the interface relaxation method. A handful of the test problems are non-linear such as the following solved on the geometry of the lower left of Figure 2. Note that there are singularities at the reentrant corners.

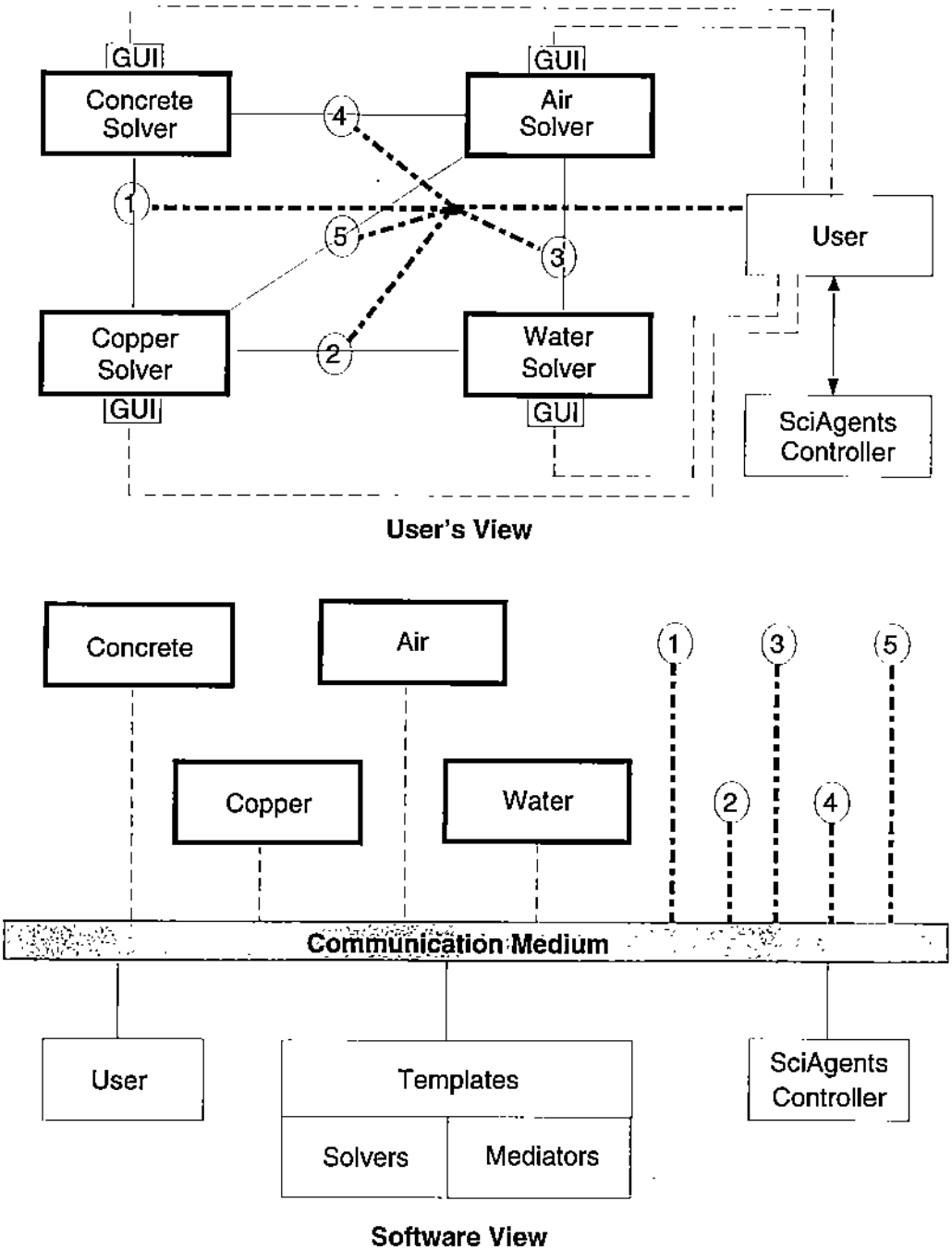

Figure 3: The SciAgents architecture as seen by the user (top) and by the SciAgents software (bottom). The solver agents are in boxes with heavy borders and the mediator agents are in ovals with numbers.

Set 1:	$UU_{xx} + (1+U)U_{yy} + aU(1+U) = b(x^2+y^2-2)$	Domain 1
	$U_{xx}/(1+(x-y)^2) + U_{yy}/(1+(4x-5y)^2) + cU/(101+U) = 0$	Domains 2 & 4
	$U_{xx} + U_{yy} - d(U_x + U_y) + cU = 0$	Domain 3
Set 2:	$U_{xx} + U_{yy} + ae^{x+y+U/500} = b(x^2+y^2-2)$	Domain 1
	$UU_{xx} + UU_{yy} + (U_x+20)U_y + 2(U_x-20)U_x = 0$	Domains 2 & 4
	$U_{xx} + U_{yy} - b(U_x + U_y) + aU = 0$	Domain 3

The total human time to solve a composite system on the domain shown in Figure 2 is about three hours. This counts the time to create the SciAgents solver, i.e., to make the sketch, to define completely four PDE problems with PELLPACK, to define the initial guess (it must be continuous along all the interfaces, and equal to external boundary conditions), to create five mediators, to select relaxation formulas along with their parameters, and to assign all the agents to workstations on the local network. The solver agents are assigned to separate workstations and everything else is assigned to another workstation. This creation effort requires about half the time. Once the composite PDE solution is started there are significant waits (30 seconds to 2 minutes) due to the time for a particular workstation to solve a particular PDE (of course, four of these could be working simultaneously). Sun SPARCstation 5, 10 and 20's are used. The solution of the particular problems being discussed takes a maximum of 53 iterations to achieve engineering accuracy; not all the solvers are executed the same number of times due to the loosely coupled, agent-oriented control mechanism of SciAgents. During this time the user adjusted the relaxation parameters some; the singularities at the reentrant corners account for some of the slow convergence. Once the global solution is obtained it then takes some time to collect, save, patch together, and view its four pieces. The number of "iterations" to obtain engineering accuracy usually ranges from 15 to 100; the number is not easily predicted in advance and is strongly influenced by relaxation parameter values. We have also solved several hundred 1-D composite PDEs using interface relaxation.

Real world applications are in progress. One is the model for a window Josephson junction in super-conducting films [1]. Let Ω_{in} be an action window region (see Figure 4) of a Josephson junction which is embedded in a global domain Ω, where $\Omega_{out} = \Omega \setminus \Omega_{in}$ is the idle region without super-conductivity. The phase difference $U(x,y)$ of the order parameter in the super-conducting films satisfies the nonlinear sine-Gordon equation inside Ω_{in} and is harmonic outside:

$$\begin{cases} \nabla^2 U_{in} = \sin U_{in} & \text{in } \Omega_{in}, \\ \nabla^2 U_{out} = 0 & \text{in } \Omega_{out}. \end{cases} \tag{1}$$

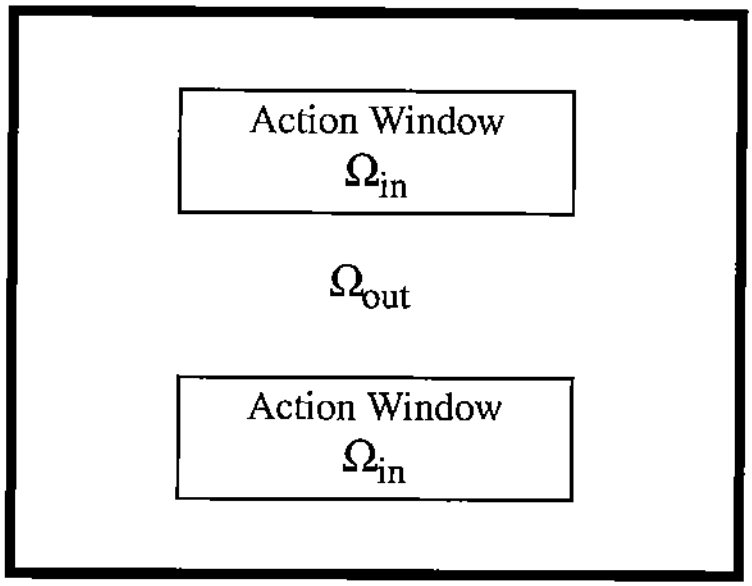

Figure 4: Typical configuration of a two window Josephson junction. The solution is specified on the external boundary and two interface conditions are specified on the internal boundaries.

These solutions are subject to the following interface and boundary conditions:

$$\begin{cases} U_{in} = U_{out} & \text{on } \partial\Omega_{in}, \\ \frac{1}{L_{in}}\frac{\partial U_{in}}{\partial n} = \frac{1}{L_{out}}\frac{\partial U_{out}}{\partial n} & \text{on } \partial\Omega_{in}, \\ \frac{\partial U_{out}}{\partial n} = g & \text{on } \partial\Omega, \end{cases} \tag{2}$$

where the surface inductances L_{in} and L_{out} are, in general, different. The domain Ω_{in} may consist of several disjoint junction windows. The shape of these windows plays an important role in the stability of the junction and oval or "bowtie" shapes are usually the best.

These experiments lead us to believe that interface relaxation has high potential as a general method for composite PDE problems. At this time there is considerable uncertainty about things like its scope of applicability, how to choose good parameters for relaxers (there are analogies with iterative methods for linear systems) and its implementation for time dependent problems.

This article and our work has not concentrated on algorithm or code performance because the principal uncertainty is whether interface relaxations works and not how fast it works. For the composite non-linear PDEs discussed above, there are no "established" alternative solution methods. Yet parallelism is an inherent property of interface relaxation and the SciAgents systems is

slowed down greatly if all the agents are placed on a single computer. Further, interface relaxation is very relevant to traditional domain decomposition; the early analyses and implementations were for solving a single PDE. One might view this method as a "continuous Schur Complement Iteration" method; in any case, it is likely to lead to new insight and ideas for domain decomposition. Our experiments have not exhibited marked differences in performance for single and composite PDE problems. The analysis of a simple model problem in [7] shows the convergence rate is good and independent of the number of domains; this is confirmed experimentally with up to 500 domains. We conjecture that its convergence will eventually be shown to be independent of both the number of domains and numerical method used for broad classes of composite PDEs.

The work reported here has been led by T. Drashansky, E. Houstis, A. Joshi, S. McFaddin, M. Mu, J. Rice, and E. Vavalis with substantial assistance from A. Catlin, P. Tsompanopoulou, and S. Weerawarana. For additional information about the SciAgents system visit its web site http://www.cecs.missouri.edu/~joshi/sciag/. Information about the Multi-model, Multi-domain Computational Methods project, which appears to have a similar approach, is available from its web site http://www.cs.odu.edu/~keyes/nsf.html.

References

[1] Barone, A., and G. Paterno, *Physics and Applications of the Josephson Effect*, John Wiley, (1982).

[2] Douglas, J., and C. Huang, An accelerated domain decomposition procedure based on Robin transmission conditions, TR 289, Department of Mathematics, Purdue University, (1997).

[3] Drashansky, T.T., *An Agent Based Approach to Building Multiple disciplinary Problem Solving Environments.* Ph.D. thesis, Department of Computer Science, Purdue University, (1996).

[4] Fritzson, R., T. Finn, D. McKay and R. McEntire, KQML - A language and protocol for knowledge and information exchange, *Proc. 13th Intl. Distributed Intelligence Workshop*, Springer Verlag, New York, (1994), 134–143.

[5] Houstis, E., 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*, **23**, (1998) to appear.

[6] Lions, P.L., On the Schwarz alternating method III: A variant for nonoverlapping subdomains, *Domain Decomposition Methods for PDEs*, (T. Chan et al., eds.) SIAM Publications (1990), 202–223.

[7] Mu, M., Solving composite problems with interface relaxation, *SIAM J. Sci. Comp.*, (1998), to appear.

[8] Quarteroni, A., F. Pasquarelli, and A. Valli, Heterogeneous domain decomposition: Principles, algorithms, applications, *Proc. Fifth Intl. Symposium Domain Decomposition Methods for PDEs*, (D. Keyes, et al., eds.) SIAM Publications (1992), 129–150.

[9] Rice, J.R., *Collaborating PDE Solvers*, slides at http://www.cs.purdue.edu/people/jrr/slides/Collaborating/index.html.