An Open Structure for PDE Solving Systems

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

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

Sanjiva Weerawarana

Report Number:
94-035

AN OPEN STRUCTURE FOR PDE SOLVING SYSTEMS

Elias N. Houstis
John R. Rice
Sanjiva Weerawarana

CSD TR-94-035
May 1994

Proc. 14th IMACS World Congress
July, 1994
Atlanta GA

AN OPEN STRUCTURE FOR PDE SOLVING SYSTEMS

Elias Houstis, John R. Rice, and Sanjiva Weerawarana[1]
Department of Computer Sciences, Purdue University
West Lafayette, IN 47907, USA
Tel: 317-494-6003, Fax: 317-494-0739, email: (enh,jrr,saw)@cs.purdue.edu

Most PDE solving systems have a closed structure. That is, they use a certain set of problems, methods, data structures, etc., that they support and all the software must adhere to the resulting standards. Of course there are systems which allow use of a general purpose programming language (e.g., Fortran) which means anything can be done. We exclude this form of open structure in this discussion even though this feature is a useful one. Even the //ELLPACK "foreign system" approach gives only limited openness, the new module must accept a complete PDE problem and return its solution. Here we consider a structure of high level PDE solving modules which is as open ended as possible. The purpose of this open structure is to allow the problem solving software structure to directly reflect the problem solving process.

We define the structure in terms of data objects and operators on them. There are five standard data objects which we call **problem objects**; the descriptions of them are at a high level and complete details are yet to be given.

1. *PDE PROBLEM.* This is the abstract or mathematical problem expressed in continuous, symbolic terms. Its data is organized as follows:

   PDE: The partial differential equation (operator).

   Domain–Interior: Where the PDE is satisfied, *–Boundary:* The boundary of the interior.

   BC: The boundary conditions equations, including initial and global conditions.

   FRAME:

   — *coordinates system and names* — *problem parameters*
   — *solution name* — *data:* constants, fixed functions.

2. *NUMERICAL PDE PROBLEM.* This is a finite problem, there are a finite number of numerical variables and equations to be satisfied. Every function and operator involved can be evaluated exactly (to within round-off) in a finite computation. Its data is organized as follows:

   Coefficients: The unknowns to be determined.

   Discrete Domain: The geometry.

   Equations: The equations to be satisfied.

   Evaluator: The function that uses the coefficients and PDE PROBLEM information to produce approximate solution values.

[1]Work supported in part by ARPA contract DAAH04-94-G-0010

3. *SPECIFICATIONS.* This data gives the requirements for solving a PDE PROBLEM.

   Accuracy: The allowed error in solving the PDE problem.

   Parameters: Actual values for parameters of the PDE problem.

   Output: The results to be returned at the end, e.g., plots, derived functions, special values.

   Resources–Time: Limits on elapsed or computer time used.

   -Money: Limits on the cost of the solution process.

   -Computer: Constraints on the computers to be used.

4. *COMPUTER.* This describes the computational environment for solving the PDE problem.

   Architecture: The list and configuration of computers available.

   Software: The libraries, systems, etc., available on the computers.

   Resources: Any limits on the resource capacities used.

5. *SOLUTION.* This is the output of the evaluator part of the NUMERICAL PDE PROBLEM. It includes evaluation of

   PDE Solution: Given any point in the PDE domain, the evaluator produces a value for the true PDE solution which is within the accuracy specification.

   Derivatives: Same as PDE solution except values may be less accurate.

The solution of a PDE is accomplished by applying a sequence of operators to the problem data. We identify two types of operators:

TRANSFORM. This class is exact (within round-off) on the data of one of the problem objects. Examples are changes of coordinates, changes of names, substitution of variables, specification of values (accuracy, parameters, machine, etc.), factorization or reordering of matrices.

METHOD. This class either approximates in some way the data of one of the problem objects or does not apply to them. Examples are replacing a domain by a mesh, replacing an accuracy by a number of grid lines, using first terms of Taylor's expansion of functions, guesses at or improvements of data values, replacing derivatives by differences, replacing money limits by time limits.

It follows that in most PDE solutions there are intermediate data objects besides the problem objects, there usually are many of them. These operators are implemented as modules in the object oriented sense, they have certain input and output which is "advertised" externally and available for use. The methods are chained together to move from the PDE PROBLEM + SPECIFICATIONS + COMPUTER to the NUMERICAL PDE PROBLEM and then to the SOLUTION. The only constraints on the solving process is that the methods be compatible, that is, the output of one method is acceptable input for the next method in the chain.

There are two observations to be made. First, and not so important, this structure easily allows transforms that create or merge problem objects. That is, a single PDE can be replaced by many. Similarly, methods can create multiple NUMERICAL PDE PROBLEMS from a single PDE PROBLEM.

Second, and more important, is that this structure does not require a unique representation of particular data items. It merely requires that methods be compatible with one another in their application. This is implemented by modules specifying not only their input/output but the types (representations) they assume for them. If multiple and incompatible representations are used for certain data items, then the PDE solving system must be rich enough in methods to make this useful. Note that incompatible representations can be used in the early parts of two solution chains (say, finite differences on a rectangular grid and finite elements on a triangular grid) and yet both chains can use the same method later (say, an iterative method for linear systems).

We illustrate this structure with a complex solution process involving three types of domain partitioning (see Figure 1) and multiple PDE problems being created and solved. The original problem on a rectangular domain has a singularity at the lower right corner and a sharp boundary layer along a curve across the domain. The subregions associated with parallel methods to solve the problem are also shown. The solution approach starts with

- Create region B by a quarter disk around the singularity. Change PDE to polar coordinates, expand solution in an asymptotic series, then discretize with a singular function basis using collocation, finally solve for coefficients with direct, full matrix solver.
- Create region C along the boundary layer and introduce an adaptive, curve oriented mesh in the region. Discretized the PDE using second order, boundary layer type finite elements.
- Define region A as the rectangle minus the quarter disk. Region C overlays A and solutions are transferred between them by
  (a) interpolating region A solutions to obtain boundary conditions for region C.
  (b) interpolating region C solution to obtain grid values for region A. Note that this partitions region A into two subregions, but this is not an essential fact.

The overall computations for the PDE solving process is outlined below.

1. Solve PDE on region A+B using a coarse grid and nothing special for the singularity or boundary layer.
2. Use this trial solution as initial guess for values along the boundaries of regions B and C.
3. Solve the PDE on region C. Break the matrix solution into 3 blocks to use a block parallel band solver for strip-like domains.
4. Interpolate solution C to the region A mesh.
5. Partition A into 12 domains and use collaborating PDE solver approach in parallel on them (plus region B). Use high order finite differences on rectangular mesh in all sub-regions except the one that touches region B. Use a special mesh there with collocation and Hermite bicubics. Solve then 12 numerical PDE problems with fixed values at the grid points in region C.
6. Solve numerical PDE problem on region B.
7. Smooth solutions along all subregion interfaces (between A+C, A+B, and subregions of A).
8. Go to Step 3.

If a flow chart (Figure 2) is made of this fairly natural process one finds about 40 boxes: there are 11 PDE PROBLEMS, 6 NUMERICAL PDE PROBLEMS, 7 subproblem SOLUTIONS, 1 final SOLUTION and many hybrid intermediate data structures/objects.

Consideration of other examples like the one above shows that this approach provides the flexibility to build the powerful PDE solving systems of the future. The next step of specification of this approach is to (a) provide the next level of objects (e.g., what are the component subobjects of the PDE operator or the computer architecture objects) and (b) give precise definitions of the interfaces between objects. It is the second step that actually defines whether the structure of the system is open or not. It is natural, especially for theoretical analyses, to specify the interfaces exactly and have the system enforce compatibility between objects. This essentially provides a global type mechanism for the system with all of its attendant advantages. However, this effectively closes the structure of the system because adding new types (data structures, problem solving capabilities, domains of applicability, etc.) has a global impact on the system and becomes too expensive to actually carry out. Further, an examination of technology outside computing shows that global typing is not used and is not necessary [1]. In programming language jargon, in other technologies one has run-time type checking and if types do not match or if objects do not recognize types presented to them, then the system proceeds to a higher level to resolve the incompatibility. The most common resolution mechanism is a TRANSFORM operator that changes types (e.g., coordinate systems, matrix formats, geometry representations). Some type conflicts are, of course, unresolvable and due to either intrinsic errors is the proposed computation (e.g., one attempts to apply a method that is not applicable) or to incomplete solving power of the PDE system. These must be resolved by the user.

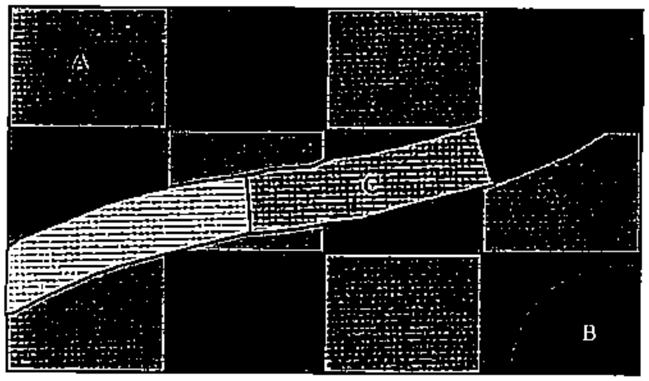

Figure 1: The application domain showing the subregions defined to (a) handle the singularity and boundary layer, (b) use 16 processors on the problem: 12 for Region A, 3 for Region C and 1 for Region B.

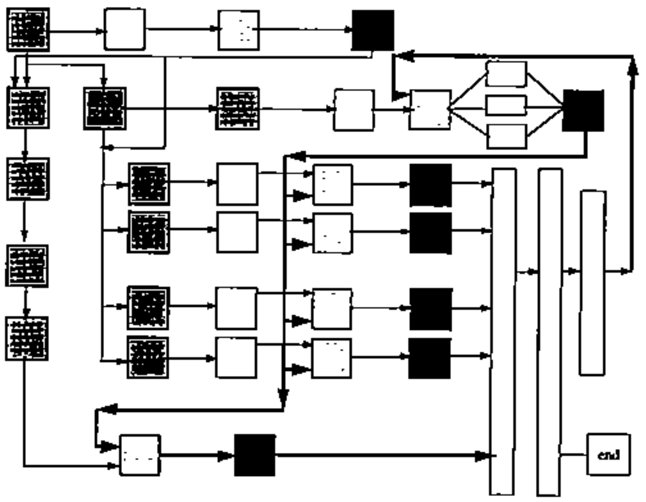

Figure 2: Flow chart of solving the application, the box shadings indicate the objects types. (The unshaded boxes are hybrid objects, very light shading is for NUMERICAL PDE PROBLEMS, medium shading is for PDE PROBLEMS, and black is for intermediate SOLUTIONS. The lines indicate the application of TRANSFORM and METHOD operators, the heavy lines the main iteration path.)

References

1. J.R. Rice and H.D. Schwetman, "Interface issues in a software parts technology," Software Reusability, Computer Society Press, 1987, pp. 96–104. Revised version in Software Reusability, ACM Press, 1989, pp. 125–139.