Object-oriented Software for Domain Decomposition Methods with Local Adaptive Refinement

S.P. Kopyssov, I.V.Krasnoperov, A.K.Novikov, V.N.Rychkov*

Institute of Applied Mechanics UB RAS, Izhevsk, Russia

Key words: software tools and environments, parallel algorithms, parallel unstructured solvers.

Recently the methods of analysis of sophisticated engineering objects (domain decomposition, domain composition, domain partitioning, etc.) are strongly developed. The main idea is to divide the main sufficiently complicated task into the tasks corresponded subdomains and then to solve them by the efficient method. That enhances speed and efficiency considerably when large and complex systems are calculated.

Domain decomposition allows complicated tasks to be solved in different fields of research. Its practice application is restrained by considerable efforts in implementation. It is common knowledge how much time is spent on creation of computing programs. They are especially far from ideal when non-professional programmers create them. Furthermore, the majority of the existing domain decomposition software is built up with procedure programming languages (Fortran, C). Therefore, these packages are difficult to use and practically non-extensible. Object-oriented programming for domain decomposition enables one to understand various aspects and properties of the developed program system, so that it essentially facilitates program developing, testing, maintenance, modification and creating of a new version.

To re-use program code, to make it extensible and flexible, ensuring object-oriented principles must be used in design of the problem solution algorithm. Following these principles, first the main tasks of the domain decomposition must be recognized. Then they are isolated in single classes, with their interfaces defined. It is important that the interfaces enable classes to work with each other and new classes to be added without changing of the existing ones. Well-designed object-oriented program system enables to conduct and change experiments with new solution algorithms and to use different ordering schemes, system of equations storage and solving methods, boundary conditions handling, etc. In domain decomposition several basic subtasks may be singled out, some of them are independent:

- To partition domain into subdomains.
- To order node degrees-of-freedom.
- To assemble matrix system of equations, taking into account element and node contributions.
- To impose boundary conditions.
- To solve system of equations.
- To update node quantities according the solution obtained.

Theoretical object analysis of the basic domain decomposition tasks has been carried out. It has resulted in detailed classifications of the objects, algorithms and computational problems arising from the domain decomposition methods. All objects of the model are sorted into four groups:

- **Modeling classes**. Classes used to create and to describe analytical model of the problem (nodes, elements, domains, boundary conditions, etc.).
- **Numerical classes**. Classes used to carry out the numerical operations and to store the data (matrices, vectors, system of equations, graphs, etc.).

* E-mails: {kopyssov, ilya, an, bobr}@udman.ru

- **Analytical classes**. Classes used to perform the analysis of the problem (solution algorithms, integration schemes, boundary condition imposition, equation ordering methods, etc.)
- **Domain Decomposition classes**. These are classes used in the analysis of the problem by means of domain decomposition method (subdomains, subdomain equation solver, etc.)

Modeling classes are used for creating analytical model of the problem, i.e. they represent elements, nodes, boundary conditions, loads, domains, etc. For the complex problem it is important to create the analytical model in simple and clear manner. *DomainBuilder* and *Domain* are the main classes of this group. Class *Domain* stores components of the analytical model and represents methods to handle them. Different variants of the analytical model editor *DomainBuilder* allow creating analytical models from the file, graphical editor, CAD system, etc.

The main class of the analytical classes subsystem is the *Analysis* class. It is an aggregation of the following objects:

- *SolutionAlgorithm.* It is responsible for the orchestrating the steps performed in the analysis.
- *AnalysisModel.* It is a container for storing and providing access to the objects of *DOFGroup* and *AnalysisElement* classes. The *DOFGroup* class represents the degree-of-freedom at the nodes and new degrees-of-freedom introduced into the analysis to enforce the constraints.
- *Integrator.* This class is responsible for defining the contributions of the *DOFGroup* and *AnalysisElement* objects to the system of equations and for updating the response quantities at the *Node* objects with the appropriate values given the solution of the system of equations.
- *ConstraintHandler.* It is responsible for handling the constraints by creating *DOFGroup* and *AnalysisElement* objects of the correct type.
- *DOFNumberer.* It is responsible for mapping equation numbers of the system of equations to the degrees-of-freedom in the *DOFGroup* objects.

Different kinds of the problem solution, in particular adaptive, domain decomposition ones and their combinations, can be included in this object-oriented model through inheritance and polymorphism. Corresponding classes are presented below.

The extra data included in *AdaptiveAnalysis* class are following: error estimations *ErrorEstimation*, refinement strategies *Refinement*, refinement indicators *ErrorIndicator*. The different improvements of solution are implemented on basis of *Refinement* class: reallocation of nodes in 2D/3D (*r-version*), local refinement/coarsening for 2D triangle meshes (*h-version*), increasing of degrees of integrated Legendre polynomials for 3D hierarchical hexahedral elements (*p-version*). *ErrorEstimation* subclasses represent a priori and a posteriori error estimations based on: error residual, interpolation type, projection type, extrapolation type, dual method. *ErrorIndicator* subclasses provide selection of domain to refine: global refinement, strategy of maximum, equidistribution, guaranteed error reduction. The object-oriented model of adaptive analysis allows building the optimal computational model with given precision and a minimum of computational costs. Domain decomposition is the most effective way to solve the problems by *h-, p-versions* FEM.

SubdomainAnalysis class is inherited from *Analysis* and corresponds the process of domain decomposition on one of the subdomains. The number of *SubdomainAnalysis* objects at runtime is equal to the number of subdomains. This class aggregates *ConstraintHandler*, *Integrator*, *DOFNumberer*, *AnalysisModel*, *LinearSOE*, *SubdomainSolver*, *SubDomain* and *DomainDecompositionAlgorithm* to execute elementary actions of domain decomposition method. For example, *Integrator* forms tangent matrix and residual vector for a subdomain; *SubdomainSolver* solves the system of equation and forms tangent matrix and residual vector for the boundary. Implementing descendants of these classes and combining them, different domain decomposition algorithms, storage schemes and solvers of system of equations can be obtained. Domain

decomposition is widely used. In supercomputing this method can be adapted to parallel computer with any memory management and node communications. Domain decomposition algorithms naturally make solution coarse-grained parallel.

Object-oriented technique helps to define objects and methods executed simultaneously and independently. The demonstration of possible parallelization on domain decomposition design is given below.

- To define the tasks using data from all subdomains and the tasks running independently. To run the main analysis object, e.g. *DomainDecompositionAnalysis, AdaptiveDomainDecompositionAnalysis*, and several subdomain analysis objects, e.g. S*ubdomainAnalysis, IterativeSubdomainAnalysis,* which are distributed objects and can perform parallel operations. Subdomain tasks are called and synchronized by the main task. To create *SubDomain* and its set of aggregated objects in separate processes as distributed objects. This step distributes the data and makes all the operations on subdomain parallel, because *SubDomain*'s aggregated objects execute them.
- To divide domain on subdomains with help of *DomainPartitioner* subclasses which uses any *GraphPartitioner* subclasses. If analysis model is relatively small, sequential partition is used, else if obtained graph has a lot of nodes or dynamic load balancing is necessary, domain partition has been parallel. In that case the subclasses of *DomainPartitioner* and *GraphPartitioner* are implemented to encapsulate any parallel library (e.g. ParMetis, Zoltan).

Some parallel distributed implementations of objects were examined on MPI and CORBA. After that parallel distributed component technique were suggested to develop object-oriented software for domain decomposition. It is based on CORBA, component model CCM, asynchronous method invocation AMI. The technique is implemented as component application service and component template library according CCM::Components standard. Distributed data and parallel processes of domain decomposition are designed with help of components.

Besides, integration of existing MPI applications is provided by this technique. The component encapsulated MPI library ParMETIS, which included a wide range of mesh partition methods, is implemented to use together with domain decomposition components.

Object-oriented model of domain decomposition and parallel distributed component technique let us develop the series of software programs for numerical experiments and verify the efficiency of algorithms. In particular, theoretical and real rates of convergence of implemented algorithms, including adaptive refinement, were compared and computational costs were evaluated.

The work was supported by Russian Foundation Basic Researches, grant 02-07-90265.