Computer Assisted Mechanics and Engineering Sciences, 3: 169–186, 1996.

Basic concepts of an open distributed system for cooperative design and structure analysis

Józef Krok, Paweł Leżański, Janusz Orkisz, Przemysław Przybylski
Computational Mechanics Department, Cracow University of Technology
Warszawska 24, 31-155 Kraków, Poland

Robert Schaefer
Institute of Computer Science, Jagiellonian University
Nawojki 11, 30-072 Kraków, Poland

(Received September 20, 1995)

Distributed programming paradigm in key stages of CAD process is proposed, as an alternative to the conventional single-computer–single-user approach. Object-oriented technology is suggested for cooperative design and implementation of large scale engineering computations. Complex ideas concerning specification and design of the new system are presented. In addition, an example of the transformation of the old CAD system to the new environment is described.

1. Why a new concept in CAD processing?

1.1. General needs in CAD

There are three general needs which have not been met by CAD processing since the beginning of this technology:

- *Computational performance requirement.* Still greater and more complicated structures are going to be analyzed, thus huge RAM space and CPU time are necessary.
- *Cooperative work requirement.* Each large scale engineering activity is conducted by many designers and expert groups which rarely work in the same place. Comfortable communication media and safe management of common data are strongly desired.
- *Compatibility requirement.* Easy collaboration of different CAD applications (computer programs) as well as running of old fashioned codes (so called "dusty decks") together with modern ones make possible multivariate design analysis and economic code exploitation.

The main objective of this paper is to propose the initial design of a system which can satisfy all the above needs taking into account progress in hardware and software engineering as well as major drawbacks of existing CAD systems. As an explication of the concept, the example of an existing system transformation to the proposed standard will be presented.

1.2. Progress in hardware and basic software technology

We have observed a significant progress in hardware and basic software in the last several years. Single CPU (Central Processor Unit) computers have achieved their limits, however. Decreasing of the processor clock cycle down to 5 ns (e.g. CRAY-C90) increases its cost so rapidly that we

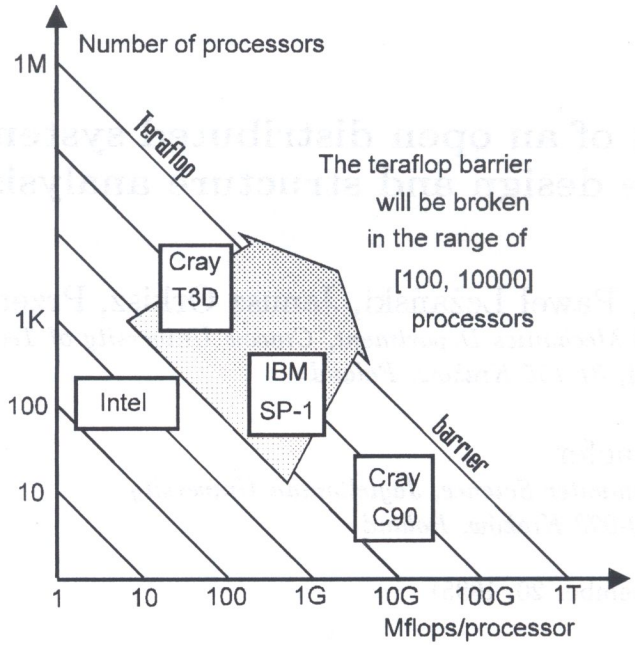

Fig. 1. High performance computer development

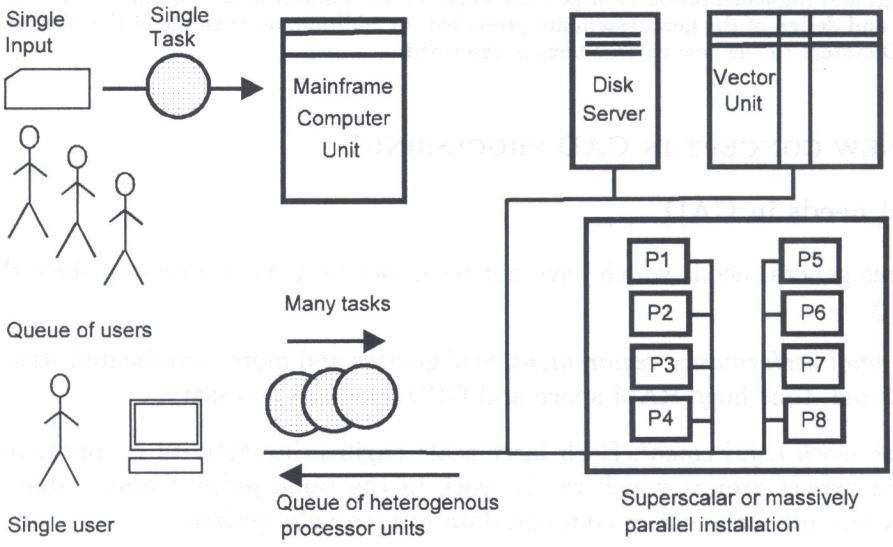

Fig. 2. User/computer environment interactions

should not expect much faster processors in the near future. The vector processors which are very comfortable for CAE computations also progress slowly due to their great price/performance ratio.

Huge performance computing may be achieved by multiprocessor installations (massively parallel, superscalar or distributed ones). First teraflop computer (10^6 MFLOPS) is expected in the range of 100–10000 processors (see Fig. 1) (cf. [19, 6]).

Fast computer networks, basic network programming and communication tools change the human/computer relationship from the old **user queue** $\leftrightarrow$ **mainframe computer** to the **user** + **powerful graphic terminal** $\leftrightarrow$ **queue of offers coming from the heterogeneous computer environment** (see Fig. 2).

Advanced distributed programming tools, such as databases and their multimedial extensions (the so called "virtual reality systems") can support multimedial communication of many users as well as safe management of and access to the common data (see e.g. [1]). Object-oriented pro-

gramming and software design paradigm increases compatibility of internal and external modules and data, due to the unified message passing interface. Therefore, they simplify the development of large scale CAD systems and may be helpful in "dusty decks" adaptation.

1.3. CAD software crisis

Unfortunately, there are only a few CAD codes which can be applied within new programming environments mentioned above. Therefore, present day CAD systems usually exhibit some significant drawbacks:

- They are based on the sequential algorithms, can be run on a single computer, allow only a single user and consequently inhibit the computational performance and designers' cooperation.
- They are difficult to rebuild and adapt to new applications.

The situation described above results from at least two reasons:

- Vendors want to protect their systems against illegal use (discordant to the license agreement) which is quite impossible in the case of distributed systems.
- Present day CAD systems are based often on frontal (or similar to frontal) technique which is strongly coupled and impossible to parallelize.

2. Modern CAD system requirements

The creation of a new concept of a CAD system should to be preceded by defining a detailed list of requirements to which the system has to conform. The paper presents them as divided into engineer-oriented and computer-oriented groups.

2.1. Engineering oriented needs

The engineering oriented needs have been arbitrarily divided into three groups according to the special requirements of computer resources necessary for their implementation. The following list, which exhausts the present CAD technology demands, is partially covered by the existing old fashioned codes completed in our group (see e.g. [7, 8, 10, 12, 13, 23, 25, 26, 27]) during the last 15 years.

2.1.1. Basic physical model requirements

- *reference frame*: total Lagrangian, updated Lagrangian, Euler description, combined Eulerian-Lagrangian and convective description,
- *type of mathematical formulation*: problems formulated by simultaneous differential equations (local description) or by variational principles (global approach); generally, problems formulated by sets of algebraic, differential and integral equations with non-equality boundary conditions (differential and variational inclusions),
- *constitutive models*: linear elastic, anisotropic, nonhomogeneous; nonlinear hyper- and hypo-elastic; elastic-perfectly plastic, elasto-plastic with hardening or softening: isotropic, anisotropic or mixed, visco-elastic, visco-plastic, high temperature creep, cracking: visco-elasto-cracking, visco-plastic-cracking, visco-elasto-plastic-cracking, others (composite cases),
- *type of structure behaviors*: static, dynamic, stepwise etc.,

- *type of loading*: concentrated loads, distributed loads with arbitrary distribution of density, axisymmetric loads, gravity loads, initial stresses and strains, initial velocity of stresses and strains, initial velocity (of generalized displacements) and accelerations, imposed non-stationary fields of strains and stresses, temperature (or shrinkage) loading, deformation dependent loading, live loading and non-conservative loading, cyclic, random, gyroscopic, non-proportional and contact loading,

- *support and/or boundary conditions and constraints*: at boundaries, axisymmetric, at internal points, prescribed displacements, sliding surfaces, support at contact points, elastic foundation, frictional supports.

2.1.2. Methodological requirements

- *methods of problem discretization*: FEM, FDM – local formulation, FDM – variational approach, mixed FEM/FDM schemes: simultaneous discretization of domain by FEM and FDM, postprocessing of FEM results by FDM formula, generation of finite elements characteristics by FDM, generation of finite difference characteristics by FEM, flexible mixing of both local and global approaches,

- *symbolic problem definitions and shape of mathematical model*,

- *types of elements* (here, "element" means finite element or finite difference star): line elements, flat elements, flat or curved elements in space (formulation on a manifold), three dimensional elements, axisymmetric solid elements, axisymmetric line elements, gap elements, discrete elements, group of discrete points,

- *symbolic generation of FD and FE operators.*

2.1.3. Special requirements

- *adoption*: methods of a posteriori error estimation: interpolation types of estimators, estimators based on duality theory, residual element- or subdomain-estimators, postprocessing type estimators, estimators based on solution obtained by other methods; strategies of new mesh density determination (definition of new diameters of elements and stars), methods of mesh refinement and/or enrichment, methods of postprocessing refinement of solution, methods of mapping of history dependent (or independent) variables from the old to the new mesh, adoptions with regard to changes of number of DOFs in nodes, number of support points, constraints etc.,

- *variable mesh in calculation process*: changes in mesh topology caused by domain modification during the simulated process, appearance and disappearance of nodes, appearance and disappearance of elements, partial or full mesh rebuilding because of strong local degeneration,

- *multilevel and multigrid approach*,

- *stepwise techniques and bifurcation analysis* (stepwise simulation, reverse loading in plasticity, viscoplasticity etc.): incremental (non-iterative), incremental iterative with constant or variable metric (Newton or quasi Newton), Newton or quasi Newton type of methods with additional constraints, e.g. different types of arch methods (with line searches) with a possibility to jump (pass by) limit and bifurcation points; like the above, but with a possibility to analyse problems in special reference subspaces (bases) — modal analysis plus 'response system' analysis; direct integration methods, etc.,

- *ill posed optimization and inverse problems*: earthen dam stability analysis and monitoring, defectoscopy, etc.,

- *substructuring*: repeated use of identical substructures, mixing linear and nonlinear substructures, mixing substructures with different types of nonlinearities,
- *restart possibility* and possibility to return to last or required equilibrium configuration.

2.2. Computer-oriented needs

The engineering requirements enumerated in the previous chapter should be supplemented by additional demands concerning the computer implementation of the new CAD systems [14]. We present them in the form of an external specification, as used in a first step of computer program design. The list below often refers to the engineering requirements previously specified. From our point of view, the major issues are as follows:

2.2.1. Data management and performance-increasing methods

- *Distributed and heterogeneous environment* — the experience of many computer centers shows that distributed systems will dominate in the next decade as more efficient and flexible than traditional supercomputer architectures.
- *Metacomputing* — a user of the system should not be aware of the existence of a network. The system should be seen as a single computer with a great amount of memory and software resources (necessary to cope with computing-intensive applications, see Section 2.1).
- *Parallel application* — the system must allow the programmer to increase performance by using parallel algorithms. Many engineering problems can be solved with divide-and-conquer technique (mesh generation, multigrid, stepwise technique, domain decomposition solver, see Sections 2.1.2 and 2.1.3).
- *Users' cooperation* — contemporary research requires involvement of many various specialists in one project. The system must be able to use the same data and applications concurrently in order to allow multiple users to work together.
- *Optimization of distributed processes* — the system should automatically decide where to run each application to assure optimal load balance for all computers (especially important in the case of parallel applications mentioned in Sections 2.1.2 and 2.1.3).
- *Project management* — simultaneous work on the same project may by difficult without advanced management. We are convinced that each CAD system must also include workflow tools to organize work in an efficient way.

2.2.2. Technical requirements

- *Environment independence* — system cannot be bound to one hardware and software environment. Rapid development of software and hardware platforms may require substantial changes of those environments in the future.
- *User interfaces* — users of the system may have different terminals. The system must provide suitable communication methods for users of graphics consoles as well as simple terminals both in the interactive and batch mode. (In particular, important in on-line control of simulation processes, e.g. adoption, stepwise techniques, see Section 2.2.3)
- *Data security* — data ought to be accessible only to authorized users. The system must support user accounts with access levels associated with specific projects.

- *Fault tolerant* — the system should support data and process mirroring which is used when reliable results are required. Additionally, transactions and roll-back mechanisms allow to keep computations running even after some nodes have crashed.
- *Plug & play* — adding new components to the system by the user should be very simple, usually limited to a registration process of a new application.

2.2.3. Multivariant analysis

- *Revision control* — a mechanism for keeping track of multiple versions of project computation; it is especially helpful in stepwise techniques and in cases of bifurcation (see Section 2.1.3).
- *Common graphics pre- and post-processing* — the same type of interface to pre- and post-processing must be designed to allow easy data exchange within the system.
- *Reusing of existing applications* — there are many very good CAD applications, usually written in old-fashioned style, which should be usable in the new system automatically, possibly without any interference with to the source code.
- *Multiple applications* — contrary to the "general solver" approach, we suggest building a system consisting of many separate applications, each fine tuned to a small range of problems, e.g. applications specialized to different physical features, different types of formulation, etc. (see Section 2.1.1).

3. Basic design concept

In the light of the requirements enumerated above, the task of creating such a system seems to be extremely complex. Indeed, it is not simple both to design and implement, but, fortunately, the new methods of developing software make the task possible.

Nowadays, the most promising software development technology is the object-oriented approach. Contrary to the general opinion, it is a very powerful tool for problem analysis and software design, not only a method of programming. Object-Oriented Analysis (OOA), Object-Oriented Design (OOD), and Object-Oriented Programming (OOP) may constitute the most efficient instruments for large systems development [2].

Object-oriented technology is also considered to be the only solution for creation of reliable heterogeneous distributed systems. The activity of OMG (*Object Management Group*) which supports the standard of transparent message passing mechanism for remote objects — CORBA [20] (*The Common Object Request Broker Architecture and Specification*) in our opinion fully confirms this statement.

Therefore, we believe that the system has to be designed with the use of object-oriented methodology. It makes possible to satisfy the requirements of openness, reuse, and multivariant analysis, and allows us to prepare a well-documented project and base its realization on existing tools for distributed programming (e.g. Arjuna [29], ORBeline [21], Emerald [5], etc.).

The most popular Booch notation [2] for object-oriented analysis will be widely used in the following parts of the paper, due to its compactness.

3.1. Object-oriented software design

The main feature of the O-O technology in software design and development is *encapsulation* of data and *adjoining* methods which implement operations on the data into a union called *object* (see [18]). Objects are active entities which can communicate with each other using *messages* containing data and commands.

That mechanism is called a *message passing technique.* In object-oriented distributed systems there are no programs in the traditional meaning of the word, that is entities which start, read input data, process it, and return results (Fig. 3). Instead, input data and a *requests* to process it are supplied in a message to an active object. When results are ready, the object sends an acknowledgement message to confirm completion of the calculations.

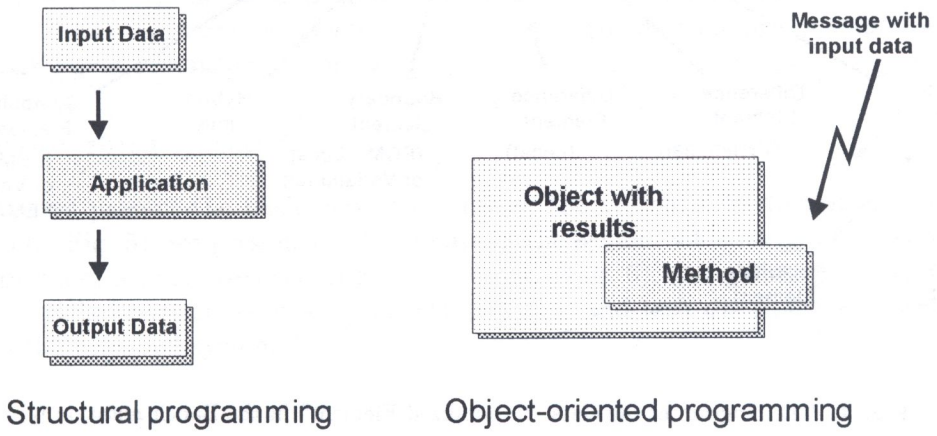

Fig. 3. A comparison of traditional and object-oriented programming

Objects can be "born" and "die" and their "parents" are called *classes.* Objects derive data structure (record type and length, etc.) and methods from their classes. Classes may be derived from *metaclasses* or other classes and this procedure may be repeated iteratively. Many O-O systems have a specific *motherclass* from which all the others may be derived. This important feature of O-O design has been called *inheritance.*

The objects (or classes) which are derived from another class are called *instances* of this class. Instantiation is perhaps the most basic object-oriented software reusability mechanism. Objects may be instantiated either statically or dynamically. Statically instantiated objects are allocated at compile-time and cease to exist after the program execution is stopped. Dynamically instantiated objects require run-time support for allocation, and also for explicit deallocation or some form of garbage collection.

As a natural realization of object-oriented technology, *object-oriented systems* were invented. They provide resources through objects, sometimes all the way down to the machine level (O-O architectures are found at the bottom). They are almost always distributed systems, allowing objects to be passed freely between machines.

In parallel to O-O systems, *object-oriented databases* were developed [30]. OODBs provide all the benefits of object-orientation, as well as the ability to have a strong equivalence with object-oriented programs. An equivalence would be lost if an alternative were chosen like in a purely relational databases.

3.2. Class "Generalized Element" as a basic class in discrete structure analysis

The idea of *generalized element* plays central role in the discrete analysis of boundary value problems. It takes its origin in earlier works of our group [11, 10] and covers features of both finite element and finite difference approaches. The object-oriented technology allows us to encapsulate data that describe the element with all the essential operations used in the FEM and FDM implementation [4]. Contrary to the structural programming which strictly separates data (usually represented as arrays or structures) and operations (represents as procedures and functions) the object-oriented approach binds them into one class.

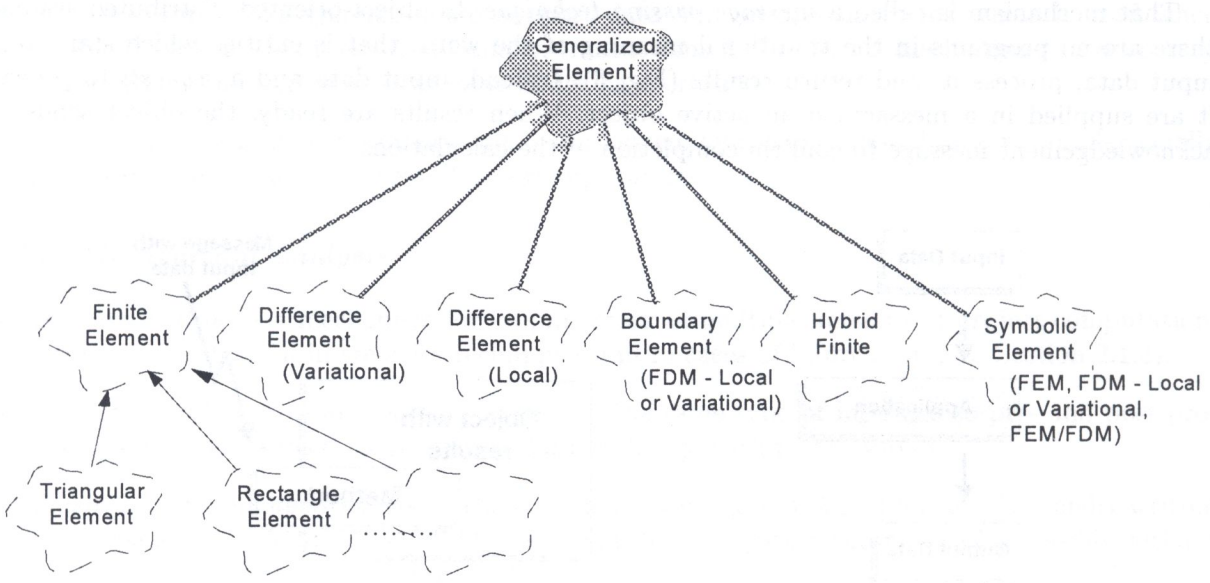

Fig. 4. The diagram of the class "Generalized Element" in the Booch notation [2]

The class "Generalized Element" (see Fig. 4) is a base (abstract) class for special, dedicated classes which consist of the following attributes and methods:

Methods

- Class of geometrical transformations (affine, polynomial etc.),
- Shape functions,
- Calculations of matrices (geometrical Jacobian, stiffness, capacity, lumped mass, consistent mass, dumping, physical nonlinearities, etc.),
- Right hand sides (initial loading vector, vector of reactions, vector of residual forces, vector of lumped mass, vector of heat loading, etc.),
- Calculation of postprocessing values (local error indicator of approximation, local error indicator of violation of physical laws, physical quantities depending on derivative or integral of the solution (e.g. stresses)).

Attributes

- Geometrical data,
- Nodes,
- Data that define degrees of freedom,
- Matrices and right hand sides,
- Results,
- Indicators of solution and physical post-processing values

3.3. O-O paradigm as a homogenous environment for CAD data/method managing

Although the object-oriented programming have been used successfully in engineering analysis of structures for solving significant particular problems (see e.g. [16, 4]), the modern object-oriented technology leads to an optimal solution of CAD processing in heterogeneous computer network.

3.3.1. Basic concept

The whole CAD system may be designed as a *class collection*. Each class represents a significant group of data and tasks, such as geometry modeling, FE/FD grid generation, solid structure analysis, graphical postprocessing etc. Main abstract classes contain basic common data features and methods for a whole group of applications. Metaclasses or classes derived from the abstract class (see Figs. 5 and 6) specify the methods and data structures for different kind of applications of the same type (e.g. *Delaunay's triangle generator class* and *Lagrange rectangle generator class* are the instances of basic *grid generation* metaclass).

3.3.2. Object-oriented structure

In this chapter we present the basic class structure of the system. The definition is divided into two levels. First (Fig. 5), we present the local part of the system which will allow us to define our engineering problem locally in space and time; second (Fig. 6), the global part of the system is presented. The global part is responsible for computation and task distribution control as well as persistence of the designed system.

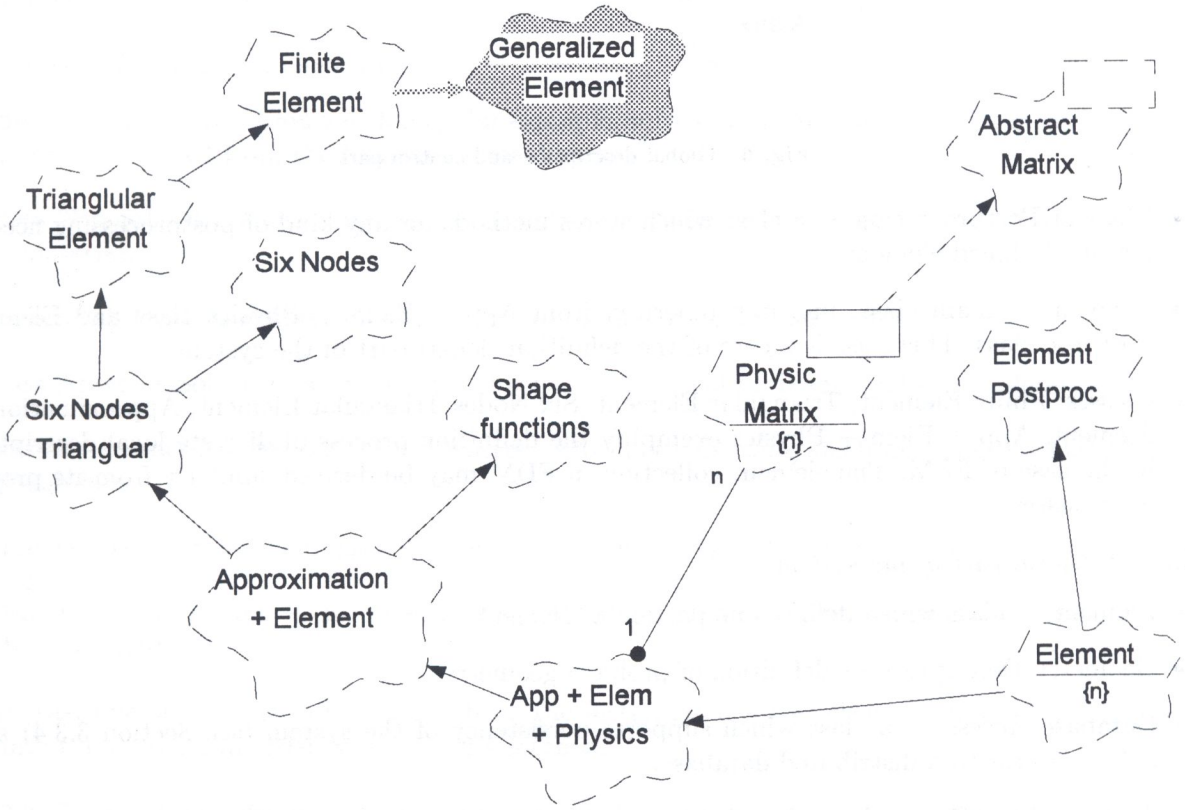

Fig. 5. Local description part

3.3.2.1. Local part of the system

- Generalized Element — basic class in discrete structure analysis (see Fig. 4).
- Abstract Matrix — template for any matrix.
- Physic Matrix — collection of instances of Abstract Matrix template. These classes are designed for definition of element physics.

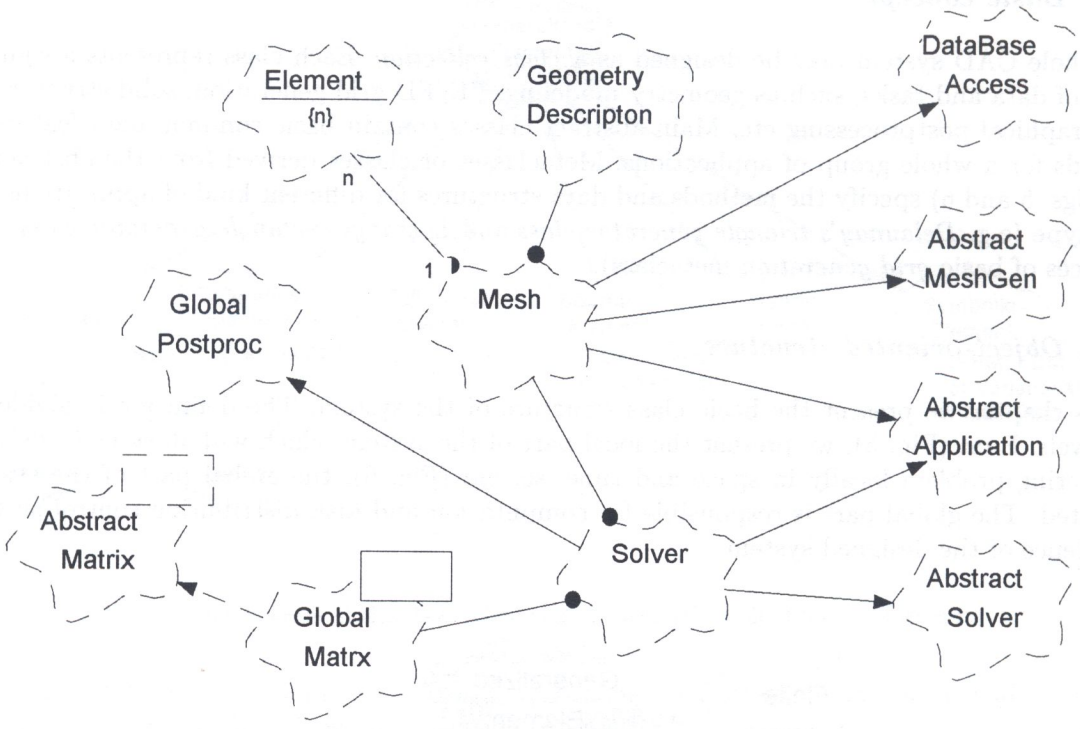

Fig. 6. Global description and control part

- Element Postprocessing — a class which stores methods for any kind of postprocessing needed for the designed element.
- Element — main class, multiply inherited from App + Elem + Physics class and Element Postproc class. This class is a core of the definition (local) part of the system.
- Classes: Finite Element, Triangular Element, Six Nodes Triangular Element, Approximation + Element, App + Elem + Physics exemplify the definition process of discrete local description in the case of FEM. The element collection in FDM may be derived similarly from its proper metaclasses.

3.3.2.2. Global part of the system

- Element — class which defines one particular element.
- Geometry Description — definition of problem geometry.
- Database Access — a class which supports persistency of the system (see Section 3.3.4) and defines access to a distributed database.
- Abstract MeshGen — base class for any mesh generator.
- Abstract Application — base class for classes which have to be synchronized and ought to communicate by the network (see Section 3.3.4). This class supports access to distributed queue and message passing.
- Mesh — mesh generator, inherited from Abstract MeshGen has a collection of Elements, uses Geometry Description, has access to database (inherited from Database Access) and can be synchronized by the network (inherited from Abstract Application).
- Abstract Solver — base class for any solver.

- Abstract Matrix — (see Section 3.3.2.1)
- Global Matrix — class which supports matrix operations needed by the solver.
- Solver — main class of the global part of the system. Class Solver, which uses class Global Matrix and class Mesh, is derived from Abstract Solver and has distribution ability by inheritance from Abstract Application.

The example in Fig. 6 shows that creating a new application means simply proposing a scheme of inheritance and adding some problem-oriented methods.

All objects instantiated from class Solver (or another application classes) are simple elementary computational tasks in our system. Each of them may be persistent and distributed because it is inherited from class Database Access and class Abstract Application.

3.3.3. Application creation methods

The previous definition of the system is rich in methods to create different kinds of engineering applications. We try to show three possible ways to derive object-oriented applications from the introduced structure.

3.3.3.1. Simple sequential application

Simple sequential application which may, for example, solve elastostatic problems for massive structures may be derived from the following metaclasses:

- class Generalized Element (and its subclasses) — for local approximation and physical features of material,
- class Mesh (and its ancestors) — for tasks of mesh generation and refinement,
- class Abstract Solver — for strategies of solving numerical problems under consideration,
- classes Abstract Matrix and Global Matrix — for methods using algebraic computations on the global data,
- class Global PostProc and class Element — in order to provide prescribed adoption technique,
- class Database Access — in order to keep, store and deliver all necessary data during solver object activity.

At least one object has to be derived from the instantiated solver class. Such an object will be put into the network in order to execute its methods.

3.3.3.2. Distributed application supported by low level programming tools

All inheritance schemes are the same as those in the previous case. Abstract Solver class may additionally provide methods for task distribution using low level distributed programming tools (e.g. PVM). Single object may be also created.

3.3.3.3. Distributed application supported by object dislocation

Also, all inheritance schemes are the same as in the first case but multiple objects constituting elementary computational tasks have to be created. The object-oriented management system (see Section 3.3.4) is responsible for optimal object relocation in a computer network.

3.3.4. The design of a distribution mechanism

Considering requirements presented previously we propose a distribution mechanism of an object-oriented system which satisfies the majority of described demands and may be implemented using one of the existing distributed programming environments.

Usually distributed systems assume that objects are stored on homogenous computers connected through very fast LAN network. Unfortunately, the CAD system proposed here differs essentially from these postulates, so it has to cope with the following additional problems:

- data transmission rates between machines may be very different and may radically change with time;
- the system may contain computers dedicated to particular tasks (vector computers for big numerical computations or graphics workstations for visualization processes);
- there are machines that can execute only a subset of all the methods found in application objects;
- tasks distribution process must be optimal, which means that system must consider the type of task to run, present transmission rate of the network, and actual and predicted load distribution over the machines.

From our point of view, the system is seen as a *distributed database* with a few *additional mechanisms* which enable message passing, process synchronisation, and optimal distribution. The proposed extension to the database is a distributed queue which may be compared to the tuple space found in e.g. Linda [31]. All calculation requests are put in a queue, and because of its distributed nature they are seen at every site.

The project provides a *task manager* process on each machine connected to the system. The task manager is responsible for monitoring the distributed queue and deciding which request can be handled by the site occupied by the task manager. To make the proper decision it has to know the present transmission rate of the network and the load balance of the machine as well as the possibility to invoke the requested method (the task may be dedicated to vector supercomputer, graphics workstation, or the source code of the method cannot be compiled on some machines).

The simplest object-oriented description of the distribution mechanism contains two classes:

- Database Access — a class which should be the ancestor of every persistent class. It provides the persistence property by direct link to the distributed database.

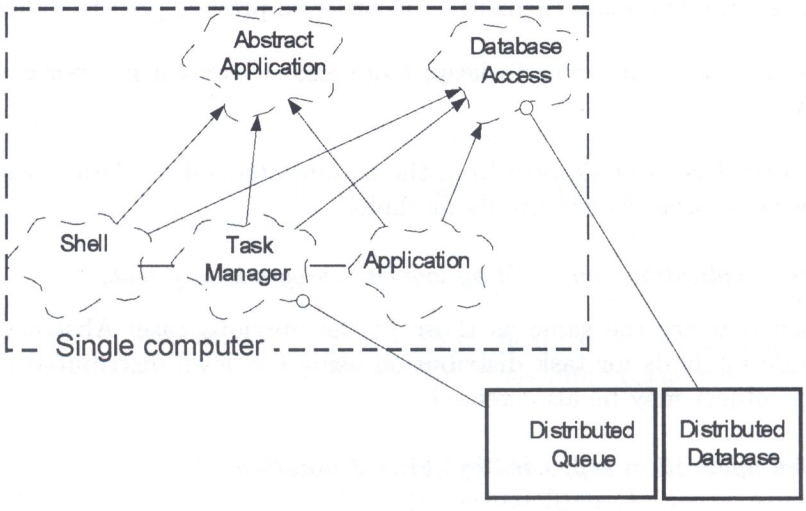

Fig. 7. Basis components of the system

- Abstract Application — a class from which every application must be derived including user interface (shell). It implements basic inter-process communication facilities.

The relationship between the Queue, the Database and all other objects is expressed in Fig. 7.

The structure of the system may provide all kinds of task distribution. It can be done by low level algorithms [24], dedicated to a task as well as by special tools which support process distribution (e.g. PVM). Additionally, the system may be connected to modern object-oriented databases which provide the object migration mechanism (e.g. EMERALD [5, 31]). Migration mechanism allows to create each object virtually (without initial allocation) and then move it to a more suitable computer. This mechanism provides very advantageous conditions for optimal load balancing which may be done globally for all objects (elementary tasks) created by the system, not only for the objects derived from the single distributed application class (e.g. distributed Solver Class).

3.3.5. Advantages of the system

The fundamental advantages of the approach described above are the following:

- *extreme level of reusability and easy application building*: only the control method has to be chosen or created, the other methods are derived form the proper metaclasses according to the presented scheme,
- many *methods of data creation and conversion* (e.g., collection of metaclasses and classes "Element") *may be defined only once*,
- objects which constitute the elementary computational tasks *have the same status* with respect to the O-O managing system for both sequential and distributed applications,
- *the optimal persistence degree* may be achieved,
- adaptation of an old application can be achieved by *object-wrapping technique* which is very easy to apply in a message passing environment. It is simply done by changing a main program into a method of the class derived from the Abstract Application. The method is invoked when a message with an input file comes and, after processing, a message with output file is sent.

4. TECHNICAL ASPECTS OF REALIZATION OF THE SYSTEM

Object-oriented systems are very popular nowadays, so a process of creating a new one has to be preceded by a thorough study of achievements made by other research groups. We indicate here which parts of modern systems can be applied in our project and what we have to take care of during the design process.

The most important operating systems which we have been acquainted with are:

- **Chorus** Micro-kernel is a distributed real-time microkernel that can be combined with the CHORUS/MiX subsystem, which is a modular, fully compatible UNIX System. There is also an O-O subsystem named COOL (CHORUS Object Oriented Layer) which provides a distributed O-O programming environment for C++. COOL supports a set of system calls that allow the creation of dynamic objects. These objects can send messages in a location transparent way, they can migrate between address spaces and sites and they can be stored in a persistent store; this is done in a transparent way, as an extension of the C++ language. Micro-kernel strategy seems to conform well with our system, especially for heterogeneous environments.
- **Spring** (Sun) [17] — is a highly-modular, object-oriented, microkernel system, with transparent remote object invocation. It is enriched by fault-tolerant and replication mechanisms. It can also emulate the UNIX system. This system is one of the most advanced future object-oriented operating systems and that is why we have to consider it in our project.

- **Apertos** — (formerly MUSE) project at Sony Researches is a meta-object based distributed OS for turning portable wireless hand-held computers into fully-connected Dynabook-like terminals; good example of computer for engineers.

So far we focused on relational databases as a database layer of our system, because of good quality and high speed of them (ORACLE, INFOMIX, etc.). But now, we are going to base our project on one or many of the object-oriented distributed databases. Existing distributed object-oriented databases as well as programming environments may radically accelerate the progress of design and implementation. Especially, we consider as important for us the work of *The Object Management Group* (OMG) which has defined *The Common Object Request Broker: Architecture and Specification* (CORBA).

The CORBA, as defined by the OMG's Object Request Broker (ORB), provides the mechanisms by which objects transparently make requests and receive responses. The ORB provides interpretability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems. The Common Object Request Broker Architecture and Specification is a self-contained response to the Request for Proposals (RFP) issued by the ORB Task Force of the OMG.

From our point of view the most interesting distributed databases and programming environments possible to adopt for our system are:

- **Arjuna** is a programming tool which supports nested atomic actions (atomic transactions) for controlling operations on objects (instances of C++ classes) which can potentially be persistent. Arjuna has been implemented in C++. The software available includes a C++ stub generator which hides much of the details of client–server based programming. Arjuna can be used to build fault-tolerant distributed applications [29].

- **ORBeline** is a complete implementation of OMG's Common Object Request Broker Architecture (CORBA). ORBeline goes beyond the standard specification to provide a SMART communication framework allowing you to easily develop large distributed applications that are robust, scalable, flexible and maintainable. ORBeline automatically picks the best communication mechanism as soon as you try to access an object and ORBeline's SMART Agent monitors communication between objects and their clients. In case of a failure, the SMART agent and ORBeline cooperate to reestablish connections between processes or their replicas.

- The **ITASCA** Distributed ODBMS is a language neutral, full-featured, active object database that supports data access from various object languages. The ITASCA database management system has features that can be found in most database systems. This includes persistent storage for data and schemas, concurrency control and locking, transaction management, multiple security levels, and logging and recovery for both CPU and disk media failure. Additional features of ITASCA include dynamic schema modification, long-duration transactions, shared and private databases, distributed version control, distributed transaction management, distributed query management, distributed change notification, object migration, and an extensible architecture.

- **MATISSE** is a distributed ODBMS based on symmetric, fine grain, multi-threaded architecture, which includes media fault tolerance (object replication), transparent on-line recovery, consistent database reads without locking, historical versioning (both schema and data objects), scalability (from few bytes to four Gigabytes for each object and up to four Giga-objects per database).

- **Objectivity/DB** has a fully distributed client/server architecture that transparently manages objects distributed across heterogeneous environments and multiple databases. It provides an application interface that uses transparent indirection to ensure integrity and provides a single logical view of all information, with all operations working transparently on any database on the network, with scalable performance as the number of users and objects increase. A higher-level

Object Definition Language (ODL) is available as well as a C functional interface, integrated C++ interface, and SQL++.

- **Versant** is a client/server object database management system (ODBMS) targeted at distributed (with full object migration features), multi-user applications. The Versant system has features belonging to typical object-oriented databases, like versioning, checkin/out, object-level locking for fine granularity concurrency control and server-based query processing to reduce network I/O.

- **Odapter** is object/relational adapter which enables object-oriented developers to share a common object model stored in the ORACLE7 relational database management system (RDBMS).

5. Example transformation of existing procedural oriented CAD system NAFDEM to object-oriented terms

Well tested and already prepared CAD codes have a big value as a computational tool but are not ready to adopt them into modern, distributed systems. We have to redesign some parts of the project (designed with procedural mode) according to object-oriented technology. The computational kernel of structure analysis programs consisting usually of high quality element libraries, integration procedures etc. may be utilized now due to the "object wrapping" technique. Here, we present the example transformation of existing, working system to new technology.

The system under consideration which covers FE/FD model was implemented as a program package called NAFDEM — "Nonlinear Analysis by the Finite Difference and Element Methods". NAFDEM is aided by a preprocessor JKJK providing the automatic execution of all needed symbolic computations (Fig. 8) like formal differentiation, derivation of various formulas or generation of source code subroutines for stiffness matrices (see [7, 10, 22, 23] for more details). Consequently, the user has only to define the form of the functional or the virtual work principle applied, to specify boundary conditions, and to describe the domain of the problem considered. The core of the NAFDEM system is a special element called DUMMY. It uses subroutines automatically generated by the preprocessor JKJK, and provides implementation of all considered FE/FD combinations in one computer program for a wide class of boundary value problems, given in the variational formulation. In that way, both the discretization of variational principles and linearization of nonlinear problems are fully automatic. All programs were implemented on a mainframe computer, and also on IBM PC AT compatibles.

5.1. Main ideas of NAFDEM transformation

- We intend to create many solvers (classes) for problem set instead of the "general" one. On-line compilation and linking is not necessary, as we may ignore blocks B1 and B2.

- We omit transformation of the JKJK symbolic preprocessor. It is not necessary to make symbolic operations on-line with computation, therefore well known modern symbolic tools (Maple, Mathematica) seem to be more comfortable.

- All the subroutines and local data structures collected in element library will wrap-in to the proper levels of *General Element* class.

- All possible strategies expressed in block-diagram C (iterations, adoption), will be put into the methods of proper *solver* classes (generated accordingly to the designed structure)

Wrapping-in process details can be expressed in Table 1.

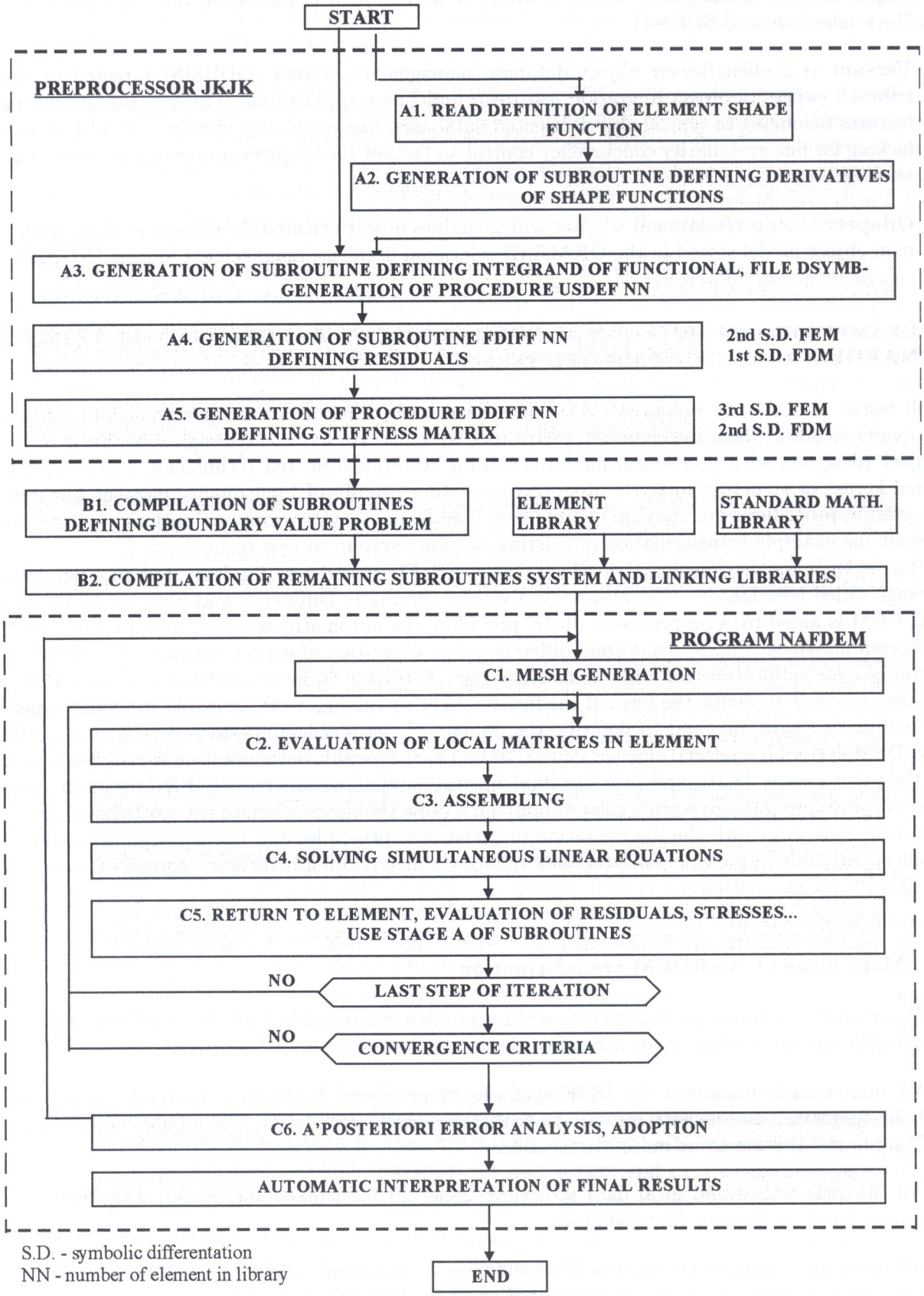

Fig. 8. Scheme of the NAFDEM system

Table 1. Scheme of the NAFDEM transformation to the O-O environment

Block	Function to be transformed	Class of proposed object-oriented structure (see. Fig. 6) which covers current function
C1	Mesh Generation	class Mesh
C2	Evaluation of local matrices in element	class Element
C3	Assembling	class Global Matrix
C4	Solution	class Abstract Matrix, class Abstract Solver
C5	Return to element, evaluation of residuals, stresses, ..., use of stage A of subroutines	class Element
C6	A posteriori error analysis	class Global Postproc, class Element
C7	Interpretation	class Abstract Application

6. Final remarks

- The object-oriented approach to complicated CAD processing (mechanical analysis of structures) constitutes a big step towards easy creation of new applications. It can be done due to the effective object-oriented analysis and software design as well as the maximal degree of code reusing, including old fashioned codes.
- The presented proposal of object-oriented environment and applications are based not only on a theoretical study, but most are the results of our research carried out in the Institute of Computer Sciences, Jagiellonian University and Computational Mechanics Department, Cracow University of Technology. The set of object-oriented tools and subsystems (e.g. graphic user interface, animation system) is called OCTOPUS [15]. The external codes created earlier by our group, NAFDEM [7, 23] and MUBS [27] are going to be adapted to the OCTOPUS standards. The obtained software system seems to be unique because of the simultaneous use of several discrete methods like FEM, FDM-local, FDM-variational together with object-oriented approach. The methods to be applied take into consideration multigridness, adaptability and ability of parallelization which constitute current directions of discrete method development.
- The environment created is a powerful tool for engineering applications design, because of extreme level of reusability, many methods of data creation and conversion, tools for persistence and distribution, and easy adaptation of old applications.
- The system has an ability for parallel and distributed processing in heterogeneous computer network. It lets users share resources and work together in one or multiple projects.
- The concept presented in this paper offers a homogenous technology to solve complicated engineering problems without any loss of efficiency in standard computer implementation.

References

[1] S. Bedford, J. Bower, L.E. Fahler, J. Marini, T. Rodde. Supporting cooperative work in virtual environment. *The Computer Journal*, **36**(8): 653–669, 1994.

[2] G. Booch. *Object-Oriented Design with Applications*, 2nd Ed. Benjamin Cummings, 1994.
[3] M.A. Ellis, B. Stroustrup. *The Annotated C++ Reference Manual.* Addison Wesley, 1990.
[4] R.R. Gajewski. An object-oriented approach to finite element programming. In: *Artificial Intelligence and Object Oriented Approach for Structural Engineering*, 107–114. Civil-Comp Ltd., Edinburgh, Scotland B.H.V., 1994.
[5] E. Jul, A. Black, N. Hutchinson, H. Levy. *Emerald User's Guide*, Technical Report FR-35. University of Washington, Seattle, Washington, August 1987.
[6] J.S. Kowalik. High performance computing. Unpublished materials from the tutorial *"New computing technologies in mechanics"*, Warsaw–Zegrze, 8–9 May 1995.
[7] J. Krok. A library of the FEM programs for analysis of nonlinear boundary problems in mechanics. *Proc. of 12-th Polish Conf. on Comp. Meth. in Mech.*, 175–177, Warsaw–Zegrze, Poland, 1995.
[8] J. Krok. Adaptive FEM in plasticity and viscoplasticity. *Proc. of First Workshop PTSK*, 124–130, IPPT PAN, Warszawa 1994.
[9] J. Krok, J. Orkisz. 3D Elastic analysis in railroad rails by the finite strip approach. *Proc. of 12-th Polish Conf. on Comp. Meth. in Mech.*, 178–179, Warsaw–Zegrze, Poland, 1995.
[10] J. Krok, J. Orkisz. A unified approach to the FE and generalized variational FD in nonlinear mechanics, concepts and numerical approach. *Intern. Symp. on Discretization Methods in Structural Mechanics IUTAM/IACM*, 353–362, Vienna, Austria, 1989. Springer-Verlag, Berlin, Heidelberg, 1990.
[11] J. Krok, J. Orkisz. A comparison between FEM and generalized FDM for solution of the problems with low continuity requirements. *Proc., of 5-th Polish Conf. on Comp. Meth. in Mech.*, Białystok, Poland, May 1983.
[12] J. Krok, J. Orkisz, J. Pietraszek. An implementation of the local finite difference method at irregular grids in the NAFDEM PC FEM/FDM system. *Proc. of 12-th Polish Conf. on Comp. Meth. in Mech.*, 178–179, Warsaw–Zegrze, Poland, 1995.
[13] J. Krok, J. Orkisz, M. Stanuszek. A unique FDM/FEM system of discrete analysis of boundary value problems in mechanics. *Proc. of 11th Polish Conf. on Comp. Meth. in Mechanics*, **1**: 466–472, Kielce-Cedzyna, Poland, 11–14 May 1993.
[14] P. Leżański, J. Orkisz, P. Przybylski, R. Schaefer. Fundamentals of an open distributed system for CAD purposes. *12th Conference CMM*, Warszawa–Zegrze, 1995.
[15] P. Leżański, P. Przybylski. OCTOPUS an object-oriented, distributed system — project of implementation. *12th Conference CMM*, Warszawa–Zegrze, 1995.
[16] D. Lucas, B. Dresser, D. Anbry. Object-oriented finite element programming using ADA language. *Proc. of ECCOMASS Conf. Numerical Methods in Engineering*, 591–598, Brussels 1992.
[17] J.G. Mitchell et al. *An Overview of the Spring System.* Sun Microsystems Laboratories Technical Report, 1994.
[18] O. Niestrasz. A survey of object-oriented concepts. In: W. Kim and F. Lichovsky (eds.), *Object-Oriented Concepts, Databases and Applications*, 3–21. ACM Press and Addison-Wesley, 1989.
[19] A.K. Noor. New computing systems, future high performance computing environments and their implications on large-scale problems. In: *Advances in Parallel and Vector Processing for Structural Mechanics*, 1–22. Civil-Comp Ltd., Edinburgh, Scotland B.H.V. 1994.
[20] *OMG Common Object Request Broker: Architecture and Specification.* Object Management Group, Cambridge, Mass, 1991.
[21] *ORBeline User's Guide.* PostModern Computing Technologies, Inc., 1994.
[22] J. Orkisz. Adaptive analysis of b.v. problems by the finite difference method at arbitrary irregular meshes — concepts and formulations. *Proc. of 12-th Polish Conf. on Comp. Meth. in Mech.*, 358–359, Warsaw–Zegrze, Poland, 1995.
[23] J. Orkisz, P. Piechnik, J. Krok. Generalized finite difference approach to analysis of plates using operator decomposition. *Proc. of 12-th Polish Conf. on Comp. Meth. in Mech.*, 256–257, Warsaw–Zegrze, Poland, 1995.
[24] G. Parrington. *Programming Distributed Application Transparently in C++: Myth or Reality?* Department of Computer Science, University of Newcastle upon Tyne, 1994.
[25] R.F. Schaefer, Z. Denkowski, S. Migórski, H. Telega. Mathematical and computational aspects of inverse problems for nonlinear filtration process. *Proc. od the Second International Symposium on Inverse Problems in Engineering Mechanics, ISIP'94*, 403–409, Paris, 1994.
[26] R.F. Schaefer, S. Sędziwy. Filtration in cohesive soils: modeling and solving. *Proc. of VIII Conf. Finite Elements in Fluids*, Vol. II, 878–886, Barcelona, 1993.
[27] R.F. Schaefer, S. SŚdziwy. Semivariational numerical model of prelinear filtration with special emphasis to nonlinear sources. *CAMES*, 1996 (in print).
[28] B. Stroustrup. *The C++ Programming Language* (2nd edition), 1991.
[29] *The Arjuna System Programmer's Guide*, Public Release 3.0. Department of Computer Science, University of Newcastle upon Tyne, 1994.
[30] J. Wilcox. Object Databases. *Dr. Dobb's Journal*, November 1994.
[31] K. Zieliński et al. *Environments for Distributed Programming in Computer Networks* (in polish). Księgarnia Akademicka, Kraków, 1994.