Mecánica Computacional Vol. XXI, pp. 3099–3109
S. R. Idelsohn, V. E. Sonzogni and A. Cardona (Eds.)
Santa Fe–Paraná, Argentina, October 2002

AN APPLICATION FRAMEWORK ARCHITECTURE FOR FEM AND OTHER RELATED SOLVERS

Santiago A. Urquiza[*], **Marcelo J. Venere**[†]

[*] Laboratorio de Bioingeniería, Facultad de Ingeniería
Universidad Nacional de Mar de Plata
Av. J.B. Justo 4302
(7600) Mar del Plata, Argentina.
e-mail: SantiagoUrquiza@fi.mdp.edu.ar

† Pladema-CNEA, Facultad de Ciencias Exactas
Universidad Nacional del Centro de la Provincia de Buenos Aires
Pinto 399
7000 Tandil, Argentina
e-mail: venerem@exa.unicen.edu.ar

Key words: Application Frameworks, Software Architecture, FEM, Finite Elements, OOP, Object Oriented

Abstract. *The basics aspects of the underlying software architecture of an Application Framework for the solution of discrete methods comprising linear and nonlinear system of algebraic equations are presented. This architecture allows the easily implementation of numerical solvers for methods like FEM, FDM, FVM, as well as solvers for others discrete methods such as Electrical and Neural Networks, etc., either for steady state or transient problems. Furthermore, it is possible to implement any method where the resulting system of algebraic equations can be build trough successive contributions of Elements -viewed as operators acting on the global matrix summing to a subset of coefficients-. The versatility of the Application Framework is based on the capabilities of a universal symbolic and numeric assembler that virtually operates with elements of any type and with an arbitrary number of nodal degrees of freedom. This is also possible due to the encapsulation of element details behind generic standardized interfaces. The design is intended to achieve high rates of reuse, specially for the main program being in addition, compliant with the principle of Inversion of Control. The only task that is expected to be performed by the user, in order to obtain an actual executable program, is to code the Elemental Matrix calculation procedure and its coupling structure. Some illustrative examples of the implementation of complex problems are provided. Its is concluded that the architecture presented here provides great flexibility on implementing and rapid prototyping special purpose solvers, with the considerable advantage of reusing practically all existing routines without introducing any changes in the main program.*

1 INTRODUCTION

The impressive advances in computer hardware occurred during the last decades have enabled engineers to challenge nature with progressively more complex and comprehensive computational models. As a consequence, the software technology has evolved increasing its complexity in such a way that the old procedural programming paradigm is becoming superseded by the more advanced software technologies like Object Oriented Programming (OOP).

Recently, a new concept has emerged from the Computer Science field, particularly from the OOP paradigm, say, the concept of Application Frameworks (AFs). AFs represent a substantial effort in the field of architectural design intended to significantly improve software reusability. A Framework is a reusable, semi-complete application that can be specialized to produce custom applications [1].

AFs reusability is a matter of architectural design based on some general principles such as Data Abstraction (DA) and Standardization of Interfaces (SI). DA stands for a set of technical procedures aimed to develop and implement generic components compliant with varying contexts. In principle, DA should capture functional and structural patterns from a class of related components. In this way, it should be possible the development of generic components suited to accommodate a wide range of potential specific implementations. Another issue that contributes to the achievement of this goal, is the definition of standard and generic interfaces. The intensive use of such interfaces allows to mix and match new components with existing ones. "New components that meet these interfaces will fit into the framework, so component designers reuse the framework design" [2].

At the same time a framework entails a description at a high level of abstraction of the objects –and their interactions- that are its constitutive parts, the same applies to the problem domain. In this way, AFs provide a general setting to interpret that context with a common vision and also with an unified language. Consequently, practitioners and developers using the same framework will have a common language to speak with each other, as they will be having the same problem scheme in mind. Thus, they also will be reusing the analysis integrated within the framework. As a consequence, productivity may be considerable improved and project cycle shortened.

Another milestone of AFs is the principle of Inversion of Control (ICP). It is a common practice to retrieve components from a library by calling them from a main program rebuilt each time a developer necessitates a new application. In contrast, the developers employing AFs reuse the main program plugging into it new components or existing ones as needed. In the former case, the programmer decides the structure of the program being responsible for control flow. In AFs, the developer's code is called by the framework code and is the framework who determines the overall structure and flow control of the application[2]. In particular, in the case of software intended to solve partial differential equations and other related discrete problems -via some appropriate numerical method like Finite Element, Finite Difference, etc- this very desirable characteristic of AFs will enhance reusability, reliability and robustness by:

- reusing some well established numerical methods to perform standard tasks. We mean for standard tasks such procedures like assembling and solving linear sets of equations, controlling nonlinear loops, time stepping, data input and output, etc.
- allowing rapid prototyping of new solvers, focusing the development and debugging efforts on a very small part of the whole code involved. Obviously, this will also enhance maintainability, and quality.

These features make possible as well the development of solvers by people who are not experts in all numerical aspects needed to tackle such a kind of engineering and science problems. In the academic context, this can be of greatest importance when teaching numerical methods like FEM or FDM to students that have little knowledge or experience in handling aspects like sparse matrix technology and other operative tasks. In this way, students are focused to the analysis of formulations from the beginning, as the only requirement is to program the code that calculates elementary matrices.

It is worthwhile to note that existing numerical Object Oriented software architectures [16][17][18] demand a deep understanding of the classes (and also their inheritance structures) that such systems provide. Consequently, in order to develop and implement specific applications using those packages, a considerable amount of previous knowledge is required from the programmers. Furthermore, several design responsibilities are transferred to them. This drawback is in part due to the architectural design of these systems is focused in offering libraries of components and numerical objects that are reused by calling them from a main program or procedure. Accordingly, existing numerical architectures for solving partial differential equations problems by FEM, FDM and other related techniques, luck from being compliant with the ICP.

Based on AFs design guidelines and emphasizing the principle of Inversion of Control, the main purpose of this paper is to present the basic aspects and data abstractions that make possible a flexible architectural design for general or special purpose numerical solvers (this architecture is an extended and improved version of that presented in Enief2000[10]). The primary goal of this architectural design is to accommodate the development and implementation of numerical software capable to deal with a wide range of engineering problems involving the use of discretized partial differential equations or some other discrete analysis procedures. Our aim is to tackle the modeling requirements for solving engineering problems (with different numerical methods such as FEM, FDM, FVM, etc.) in various areas such as structural mechanics, fluid mechanics, heat transfer, electromagnetics and also coupled problems, like fluid-structure interaction, either transient, steady, linear or nonlinear, while using the same main program for all of them. The key idea of the proposed architectural design is to meet the demands of any new problem plugging different elements into an existing general purpose program, which will remain unchanged being compliant with the ICP.

In order to enhance the clarity of the subsequent exposition we will restrict ourselves in using only the terminology of the Finite Element Method. This is because FEM has become a method specially suited to combine different structural components in the same model and also by being a method that provides great flexibility to deal with mixed and coupled problems with varying degrees of freedom.

2 DESIGN REQUIREMENTS

In this section we list the design requirements that define the main aspects the architectural design of the AF for FEM like problems must meet. The goal is to provide the AF with a number of features that make it an efficient, flexible and versatile tool for the development of these class of solvers.

The requirements for the AF are as follows:

- To deal with almost any problem that can ultimately be represented by a system of linear or nonlinear algebraic equations or a sequence of them, without being specially concerned with the specific problem formulation.
- to deal with different problems by means of attaching only new elements to it, while keeping the main program unaltered and without affecting numerical efficiency.
- able to accommodate an arbitrary number of generic degrees of freedom or kinds of unknowns
- generic implementation of boundary and other restraints conditions
- performance requirements in memory and execution time similar to special purpose programs
- to implement a variety of linear equations solvers that can be mixed or combined in different ways as necessary for the specific problems.
- Self explanatory user interfaces and data structures exposed to them handled with easy.

We have borrowed from the FEM the key ideas and concepts to construct the main DA that are the conceptual building blocks of the AF. FEM was conceived in its early moments as a matrix method, i.e., the complex behavior of a structure was represented by a global matrix which in turn was constructed by the summative contributions of several Elemental Matrices embodying the simple behavior of small pieces of the structure (Elements)[6]. The relevant point here, is the idea that the global system of equations is assembled element by element allocating some coefficients of a small (generally dense) element matrix (representing some coupling relations among a reduced number of unknowns) onto a global large (generally sparse) matrix. An

assembler procedure is responsible for this task, as well as, for the correct mapping between the local contiguous unknowns' numeration and the global one (see also [3] for further discussion about this issue). It is interesting to note that this is a extremely general procedure that can be extended to several contexts. If we provide a program with a standard way of treatment for the element matrices and with a general purpose assembler, we will be very close to achieve our goals. Of special concern is the handling of Dirichlet conditions and other restraints implementations that breaks out this general process, evidencing the necessity of specific data structures and procedures to deal with them. This is mainly due to the constraint conditions are treated at the global matrix level, breaking the conventional element by element assembling process. As a consequence, the overall program suffers a lack of structuring and generality. In this sense, it will be helpful to push back this issue to the element level, to gain in modularity and extensibility. In this form, we arrive to two additional requirements for the framework architecture:

a) that the global matrix must be constructed within an unique procedure, similar to the element by element assembler method, and no additional (a priory or a posteriory) special treatment will be needed for the treatment of constraints conditions nor Dirichlet or Neumann conditioning will be considered out of the usual elements' loop.
b) that both the numeric and symbolic assemblers be generic enough to accommodate virtually any kind of elements.

The fulfillment of these last two requirements have important implications on the capabilities of the AF. Make it possible to tackle practically any kind of problem by simply plugging new elements into the framework. Furthermore, these features transforms an architecture in that of a framework, i.e., allowing the reuse of the main program while solving problems of very diverse nature.

3 BASICS DATA ABSTRACTIONS

As was mentioned above, the architecture in based on the usual entities arising within the MEF context, such as Elements, Nodes, Degrees of Freedom (Dofs), etc. Moreover, because of is usual in this method to combine in one analysis several types of elements of diverse nature, dimension and different kinds of Dofs (truss, beam, solid elements, etc.), the FEM is properly posed as a template for a general purpose framework. So, in the followings we present certain DA that are straightforward generalizations of simple concepts of MEF, sometimes oversimplified in order to gain generality.

3.1 Data Abstraction 1. The Node

It is an entity that has attached or associated in an unique manner a certain number of unknowns or degrees of freedoms. Additionally, it can have associated some geometric information (such as coordinates). A common structure for all the nodes existing in a problem can be attained, if we associate to each one an array of length equal to the maximum number of Dofs associated with a given node (DOFT=maximun number of dofs associated with a node). In this way, each node will have an array identifying the presence or not of a particular Dof. The union of this arrays in a matrix for all the nodes conforms the Destination Array (ID) of the reference[5] or the Map Object in [4]. Here DOFT is problem dependent and the range of values are as follows:

$$\mathrm{ID[Nod]} = [\, Dof_1, Dof_2, Dof_3, \ldots\ldots, Dof_{\mathrm{DOFT}}] \tag{1}$$

With Nod =1, ...NodT (the maximum number of nodes in a problem)

$$Dof_I = \begin{cases} 1 & \text{a normal unknown is present} \\ -1 & \text{a Dirichlet unknown is present} \\ 0 & \text{the unknown does not exist} \end{cases} \quad (2)$$

The ID Array is used to map nodal unknowns or Dofs to global matrix rows and columns.

3.2 Data Abstraction 2. The Element

It is represented by a set of nodes, their corresponding Dofs and prescribed coupling relations among them. As usual, the element has an Element Type Identifier (ID) and a Material ID (a set of parameters varying element by element indeed). From the point of view of its functionality the element is responsible for providing the necessary data in order to the Archetypal Element (see below) that surrogates it, can construct the Numerical Elemental Matrix.

3.3 Data Abstraction 3. The Archetypal Element

The Archetypal Element (AE) embodies the mathematics of the formulation. It surrogates the elements in the sense that it is responsible for filling the Elemental Matrix and also it "talks" with the assembler in order the Elemental Matrix be assembled.

Additionally it has some data structures as stated below:

CommonPar[]: a list of parameters defining a set of properties common to a group of elements

IsAdditive: a property that identifies if the AE has an additive (True) behavior or not (False)

Symbolic Elemental Matrix[,]: an array of integer values that express the coupling relations between the element's Dofs. It may be specified with the input or by a private method. Its values are as given in the following equation:

$$SEM_{nm}^{de} \begin{cases} \neq 0 & \text{a coupling is expresed} \\ = 0 & \text{a coupling does not exist} \end{cases} \qquad n, m = 1, ..., \text{nodes of the AE} \; ; \; d, e = 1, ..., \text{DofT} \quad (3)$$

The rows and columns are mapped with the subsequent rule:

$$\text{Row} = (n - 1) * \text{DOFT} + d \quad ; \quad \text{Col} = (m - 1) * \text{DOFT} + e \quad (4)$$

If any of the diagonal coefficient of *SEM* is set to a negative value, it means that the corresponding row of the system of equations will be marked as Non-Additive and also the property IsAdditive will be set to False.

Numeric Elemental Matrix[,]: it represents the usual element matrix of FEM, expressing certain numerical contributions to the global system of equations. Additionally, it also comprises a column vector that is used for the contributions to the second member of the system. It is important to note here that the procedure for filling this matrix is the only code the user has to program in order to implement a new problem. For this job, it will be having at hand a set of auxiliary methods and functions aimed to facilitate this process, i.e., shape functions, jacobian calculators, gauss points tables, etc. It has the same structure that *SEM* but with real values.

At this stage we proceed to explain how the Additive property of rows works. Usually, in the FEM a row of the global system matrix is the resultant of the contributions of all the elements that has the corresponding degree of freedom in connection with it. Each element that has the specified Dof contributes adding to the coefficients of the row. In this sense, we will name this row as being additive, in other words, all the elements connected to it have the right to add to it, to make their contributions. So as to facilitate the implementation of special conditions in the same loop as for standard elements, such as Dirichlet Conditions and other types of constraint equations, we have developed the concept of Non Additive Elements and rows. A Non Addtive Element (the AE in fact) has at least one of its rows marked as Non Additive (e.g., the row that represents the Dof that has a Dirichlet Condition). This property is transferred to the corresponding row of the global system,

in such a way the numeric assembler will allow contributions to that global row only to those Non Additive elements (prop. IsAdditive=False) that have the corresponding local *SEM* row marked as Non Additive. For example, if a node has a Dirichlet condition in the 2nd degree of freedom, an element having only one node may be implemented with the corresponding SEM as displayed in next formula:

$$[SEM] = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5}$$

The symbolic assembler will recognize the -1 as a marker for a Non Additive row and consequently, it will mark the corresponding global row with that attribute. When the numeric assembler try to add a coefficient to this global row coming from a conventional element, it will be impeded as the local row of the corresponding SEM will not be marked as Non Additive. In this way, the only element that will be able to add to that row will be our "one node" Dirichlet element, putting a one on the diagonal an the corresponding value of the Dof on the second member. Is in this way as the AF architecture implements the Dirichlet conditions and other more complex restrain equations within the conventional assembling loop. Additionally, Neumann like boundary conditions are implemented as additive elements that contributes only to the second member.

3.4 Data Abstraction 3. The SubStep Object

The solution of a problem is conceived as a sequence of steps. Within each step the values of the unknowns corresponding to the problem's Dofs are obtained. The sequence is ordered by a continuation parameter (usually the time) which takes ever increasingly values each time. However, from an architectural point of view, one or more steps it makes no difference. In contrast, assembling a whole step in an unique matrix lacks for efficiency on implementing Subdomain iterations and other splitting methods. This imposes the necessity of introducing Substeps. Consequently, each substep actuates on a subset of the unknowns updating them for the next stage. In this way, a considerable improvement in efficiency is achieved on implementing multistep, fractional, subdomain and other splitting methods, avoiding strongly non symmetric matrix structures and memory duplications. Moreover, the best efficient method of solution can be chosen within each substep, selecting it from a variety of available options such as direct solvers (factored once at the beginning or each time the solver is called), iterative methods with the corresponding preconditioners, etc. In turn, this allows the use of shared memory when possible and a great flexibility to implement the more efficient sequence of numerical solution. Additionally, nonlinear iterations are permitted within each substep. Consistently with the concepts stated previously, the Substep Object structure consists in a set of properties used to control nonlinear iterations, the method of solution and the links among elements and AE, jointly with methods to implement the symbolic and numeric assemblers and the solution processes.

Up to this point, we have identified the main Data Structures that describe the AF's architecture. It is important to note that this architecture in non dependent on the particular language used to implement it. Although we have programmed it on procedural Fortran 90, a slightly better mapping between architectural structures and software objects may be achieved within the context of OOP, as well as modularity may result enhanced[3].

4 IMPLEMENTATION EXAMPLES

In this section, we briefly describe two illustrative examples in order to show the versatility of the proposed

architecture to embody some complex engineering and scientific problems.

4.1 Example 1: Mixing FEM with FDM

This working example is intended to show the ability of the Framework to easily implement mixed formulations combining Finite Elements and Finite Differences. The Poisson equation with constant source is solved for a square domain with homogeneous Dirichlet conditions. By the symmetry of the problem only an octave part of the domain is considered. As it is well known, Neumann boundary conditions are simpler when treated with FEM. In the other hand, FDM produce a matrix with minimum coupling. In this way, for all interior nodes a FDM cell with the topology and MSE structure shown in fig.1 is considered. Only the first row of the corresponding SEM is necessary, others non zero entries in the first column are set to achieve symmetry as needed by the symbolic assembler (the remaining zero entries are depicted as blanks). Observe the negative value in the diagonal defining the non additivity of that row and also that is a 5x5 matrix, where 5 is the number of nodes in the computational Finite Difference Cell. A strip (blue colored) of finite elements all around the contour are considered to provide equations for Dofs pertaining to nodes on the symmetry lines and also to easily implement Neumann Boundary conditions thereof. The resultant isolines can be observed in fig. 2 conforming the proper boundary conditions.

$SEM_{FemEl} =$

1	1	1		
1	1	1		
1	1	1		

$SEM_{FDM\ cell} =$

-1	1	1	1	1
1				
1				
1				
1				

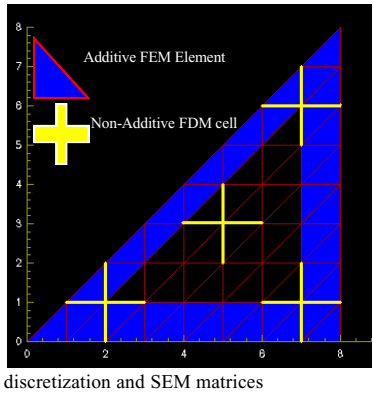

Figure 1. Example 1: Domain discretization and SEM matrices

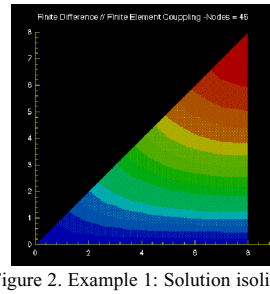

Figure 2. Example 1: Solution isolines

4.2 Example 2: Complex hæmodynamic problem

In hæmodynamic problems, it is often crucial to model zones of diverse dimensionality[7],[8],[9]. Detailed multidimensional models focalized in a small zone are complemented with one dimensional models of the arterial tree, in order to provide the former with the appropriate boundary conditions. As the problem of arterial blood flow in compliant vessels comprises the fluid-wall interactions, a complex nonlinear problem takes place. Consequently, several elements are needed to be employed to model such a multifaceted issue. Here we provide the list of elements needed:

a) Navier-Stokes elements with ALE formulation for multidimensional blood flow zones (see [13] for details)
b) Elastic Solids elements to represent the arterial wall
c) Condensed 1D Navier Stokes equations for compliant vessels to model the extended arterial tree (more details can be found elsewhere [11][12])
d) coupling elements between a) and c) (see also [13])
e) coupling elements between a) and b)[14].

In the next figures we show some results related to a problem of this kind. As we are concerned we the AF architecture, we present these results as a testimony of a straightforward implementation of a complex coupled problem within this programming environment. The results correspond to an arterial stenosis located at the Carotid Artery modeled axisymmetrically. The former bidimensional model is embedded in a one dimensional one of the whole arterial tree (see references indicated above for further details).

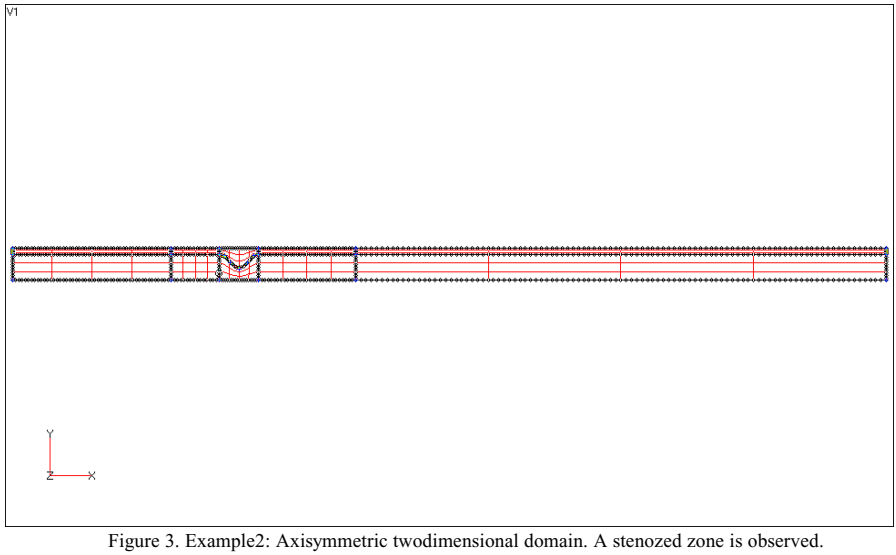

Figure 3. Example2: Axisymmetric twodimensional domain. A stenozed zone is observed.

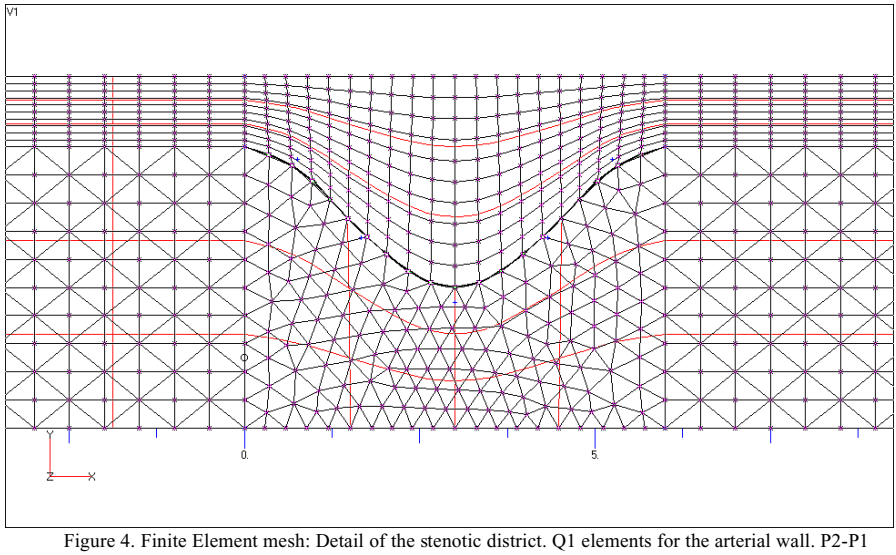

Figure 4. Finite Element mesh: Detail of the stenotic district. Q1 elements for the arterial wall. P2-P1 triangles for the blood region.

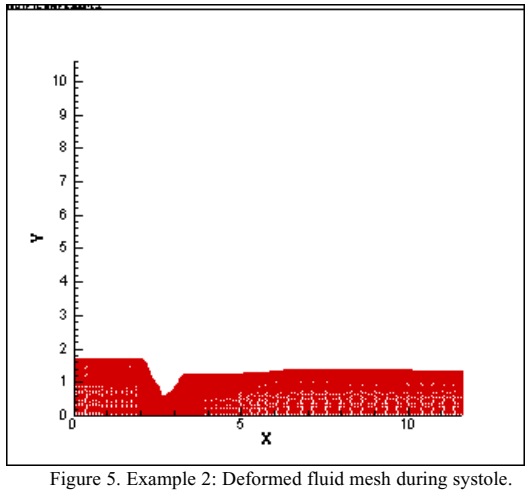

Figure 5. Example 2: Deformed fluid mesh during systole.

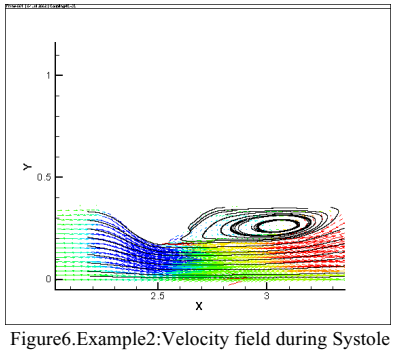

Figure6.Example2:Velocity field during Systole

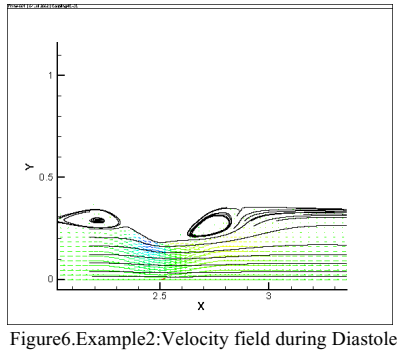

Figure6.Example2:Velocity field during Diastole

5 CONCLUSIONS

The plain architectural design proposed in this work has shown excellent capabilities to easily implement numerical solvers that combine different methods, formulations and/or elements while preserving efficiency.

The necessary tasks to put into practice a new solver are extremely abridged, demanding developers to merely implement a routine or function that fills the element matrix to be used.

Excellent reutilization levels are achieved for the main program, accomplishing the inversion of control law as a consequence of the extensive use of generic data types, standard interfaces, jointly with a modular design.

6 REFERENCES

[1] Johnson R.E., Foote B., "Designing reusable classes", *Journal of Object-Oriented Programming*, **1** (5), 22-35, (1988).

[2] Fayad M.E, Schmidt D.,Johnson R.E. *Application Frameworks. Object-Oriented Foundations of Frameworks Design*, Wiley, (1998).

[3] Lu J., White D., Chen W.F. "Applying Object Oriented Design to Finite Element Programming". *Applied Computing:States of the Art & Practice,Proceedings of the 8th ACM/SIGAPP Symposium on Applied Computing*, (1993).

[4] G. C. Archer, G. Fenves, and C. Thewalt "A new object-oriented finite element analysis program architecture", *Computers & Structures*, **70**, 63-75, (1999).

[5] Cook R.D., Malkus D.S. & Plesha M. E. *Concepts and Applications of Finite Element Analysis*. Third Edition, 49-50, Wiley, New York, (1989).

[6] O.C. Zienkiewicz and R.L. Taylor, *The finite element method*, McGraw Hill, Vol. I., 1989, Vol. II, (1991).

[7] Formaggia L.,Gerbeau J.F., Nobile F., Quarteroni A., "On the Coupling of 3D and 1D Navier-Stokes Equations for Flow Problems in Compliant Vessels", *Comp. Meth.App. Mech & Eng.*, to appear,2000.

[8] Alfio Quarteroni, Massimiliano Tuveri and Alessandro Veneziani, Computational Vascular Fluid Dynamics: Problems, Models and Methods, *Computing & Visualisation in Science*, **2**, 163-197,(2000).

[9] Formaggia L.,Nobile F., Quarteroni A., Veneziani A., "Multiscale Modelling of the Circulatory System: a Preliminary Analysis", *Comput. Visual. Sci.*, **2** (2/3), 75-83, (1999).

[10] S. Urquiza, M. Vénere. "Un framework para la implementación de métodos discretos en problemas en derivadas parciales", *ENIEF2000, XI Congress on Numerical Methods and their Applications*, Nov.2000, Argentina.

[11] S. A. Urquiza, M. J. Venere, F.M.Clara and R.A.Feijoo, "Finite Element (One-dimensional) Hæmodynamic model of the Human Arterial System", *European Community on Computational Methods in Applied Sciences and Engineering*, E. Oñate, G.Bugeda & B. Suarez Ed., Artes Gráficas Torres S.A., Barcelona, Spain,(2000).
[12] H.J. Desimone, S.A. Urquiza, M.E. Goñi, R.L. Armentano, F.M.Clara- "Simulación Computacional del Pulso Sanguíneo con un Modelo Elástico", *Revista FAC-SABI*, **3**, No3, pg 11-18 , (1998).
[13] Santiago Urquiza , Alejandro Reutemann, Marcelo Vénere y Raul Feijoo, "Acoplamiento de Modelos Unidimensionales y Multidimensionales para la Resolución de Problemas Hemodinámicos", *ENIEF2001, XII Congress On Numerical Methods And Their Applications*, Oct.31–Nov.2, 2001, Córdoba - Argentina
[14] Buffoni M., Leofanti G. "Resolución de Problemas de Acoplamiento Fluido Estructura Aplicados a la Hemodinámica", Tesis de graduación, Facultad de Ingeniería, Univ. Nac. de Mar del Plata, (2002).
[15] Are Magnus Bruaset, Hans Petter Langtangen. "A Comprehensive Set of Tools for Solving Partial Differential Equations: Diffpack", in *Numerical Methods and Software Tools in Industrial Mathematics*, M. Dæhlen and A. Tveito (eds.), 63-92. Birkhäuser, (1997).
[16] The PETSc Package WWW home page. (URL: http://www.mcs.anl.gov/petsc/).
[17] The Diffpack WWW home page. (URL: http://nobjects.com/Diffpack/).
[18] MOUSE: An object oriented framework for finite volume computations on unstructured grids. WWW home page. (URL: http://fire8.vug.uni-duisburg.de/MOUSE/).