Phenomenon Computational Pattern: coupling relationship between phenomena on multi-physics simulation

Felix C. G. Santos
José M. A. Barbosa
Eduado R. de Brito Jr
Federal University of Pernambuco
Department of Mechanical Engineering
Rua Acadêmico Hélio Ramos, s/n - Recife - PE 50740-530 - Brazil

Abstract— **Simulation of natural phenomena, such as the evaluation of temperature and air flow distribution inside a room, evaluation of material damage due to mechanical and chemical loads, are crucial in the daily life of engineers. This paper presents the Computational Phenomena Pattern, whose objective is to standardize - through a computational abstraction - the complex interaction of coupled natural phenomena in the context of the Finite Element Method. The pattern makes it intuitive and easier the representation of data sharing and dependence between different interacting phenomena when developing simulators based on the Finite Element Method. The pattern models natural phenomena data, the involved processes and their relationships, which are used in a typical simulation process.**

Keywords— **Finite Element Method, Multi-physics simulation**

I. Introduction

Computational Mechanics has had a profound impact on science and technology over the past three decades. The Computational Mechanics software industry generates several billions of dollars per year [1]. The success of Computational Mechanics is due to its effectiveness in solving problems that interest society and in providing deeper understanding of natural phenomena (facts, which occur in nature, like motion of material points and heat transfer in a continuum). Computational mechanics has a tremendous predictive power, making it possible the simulation of complex natural events and the further use of these simulations in the design of engineering systems [1]. However, in many aspects, it still applies software engineering development techniques related to the seventies. The consequence is that there are few reliable and effective tools for simulators development support and new developments in numerical methods become difficult to introduce in the existing simulators. This work is part of an effort to overcome this difficulty in which concerns to the development of an environment - called PLEXUS - dedicated to the automatic development of multi-physics and multi-scale simulators based on the Finite Element Method. We define multi-physics as a qualifier for a set of interacting phenomena, in space and time. These phenomena are usually of different natures (deformation of solids, heat transfer and electromagnetic fields) and may be defined by different scales of behavior (macro and micro mechanical behavior of materials). A multi-physics system is also called a coupled phenomena system.

The Finite Element Method is a technique for the discretization of phenomena, whose behavior is determined by variables defined on a continuum (space and time). For instance, assume that a given phenomenon is defined by a function $u : \Omega \to \mathbb{R}$, $u \in \mathcal{H}$, where $\Omega \subset \mathbb{R}^n$, $n \geq 1$, is a bounded geometric domain and $\mathcal{H}$ is an appropriate space of functions. Suppose that u satisfies a behavior law (a partial differential equation and boundary conditions, for example) $\mathcal{L}(u) = f$, where $\mathcal{L} : \mathcal{H} \to \mathcal{Y}'$ is a differential operator and $f \in \mathcal{Y}'$ is given. $\mathcal{Y}$ is a vector space and $\mathcal{Y}'$ is its dual space. Based on that PDE, an equivalent integral formulation (weak form) is then obtained through a bivariate form $B_\Omega : \mathcal{H} \times \mathcal{Y} \to \mathbb{R}$ such that $B_\Omega(u,v) = L_\Omega(v)$, for all $v \in \mathcal{Y}$, where $L_\Omega : \mathcal{Y} \to \mathbb{R}$ is a linear functional. This is the initial setting for the finite element discretization techniques.

Basically speaking, the major processes, which composes a simulation by the FEM are:

a) **Mesh generation process**: the building of an approximation, Ω_h, to the exact geometric domain Ω. Ω_h by the union of a set, τ_h, of closed simple geometric entities (edges, triangles, quadrilaterals, tetrahedrals, hexahedrals). τ_h is called the geometric mesh. Each simple geometric entity $e \in \tau_h$ is called a finite element.

b) **Discrete weak form process**: finite dimensional spaces S_H and S_Y are defined in such a way that an approximation to the original weak form can be defined as $B_{\Omega_h}(u_h, v_h) = L_{\Omega_h}(v_h)$, for all $v_h \in S_Y$, where $u_h \in S_H$ is the approximation to the exact solution u; $B_{\Omega_h} : S_H \times S_Y$ and $L_{\Omega_h} : S_Y$ are approximations to B_Ω and L_Ω, respectively. u_h is defined by a finite set of parameters $\mathbf{U}$, which are the basically unknowns of the problem. The discrete weak form represents a system (called global) of algebraic equations in $\mathbf{U}$. If the system is linear in u, then the resulting global system will be a linear algebraic system, defined by a matrix $\mathcal{K}$ and a vector $\mathcal{F}$. In this case, it can be shown that $\mathbf{K}$ and $\mathbf{F}$ can be assembled from small matrices and vectors, which are computed based on restrictions of B_{Ω_h} and L_{Ω_h} to each finite element $e \in \tau_h$. Even when the problem is nonlinear, the solution process usually consid-

Proceedings 20th European Conference on Modelling and Simulation
Wolfgang Borutzky, Alessandra Orsoni, Richard Zobel © ECMS, 2006
ISBN 0-9553018-0-7 / ISBN 0-9553018-1-5 (CD)

ers the solution of linearized systems. Therefore we may consider that each phenomenon contributes with a certain number of small matrices and vectors, which are computed at the element level.
c) **Solution algorithm process**: it means one among many ways of performing the desired simulation, that is, solving the global algebraic system of equations, obtaining the real vector $\mathbf{U}$. It is important to collect the set of all ways a solution algorithm uses the matrices and vectors computed by a phenomenon. This set can be used in the definition of a standard interface between solution algorithms and phenomena during a simulation.
d) **Visualization process**: this process comprises the computation of quantities to be visualized and the visualization procedures itself. The data of interest to the user may not be the output itself (vector $\mathbf{U}$, for the example above) of the solution procedure, but a function of it. Also, the format and data size of the solution output may not be compatible with the software used in the visualization. In those cases the quantities to be visualized are obtained by a posterior process, which should be an important part of the simulator.

If the example described above is a part of a multi-physics system, it means that B_Ω and L_Ω may be dependent on vector fields from other phenomena. If that is the case, the computation of small matrices and vectors at the finite element level (item (b) above) depends on data from other phenomena - that is, it is coupled to other phenomena. Other type of data dependence is the case where two or more phenomena are defined on the same geometry component and share the geometric mesh.

The Phenomenon Pattern is restricted to the representation of data and processes, which can be computed at the finite element level. It does not deal with abstractions neither for the Solution Algorithm Process nor for the Visualization Process. Therefore it is concerned to:

i) The ways in which the discrete weak form components B_{Ω_h} and L_{Ω_h} restricted to a generic finite element can be computationally represented. This means that the Phenomenon Pattern provides abstractions such that: (a) all data pertinent to the computation of the small matrices and vectors at the finite element level can be represented; (b) all data dependence and sharing with respect to other phenomena can be taken into consideration in a standardized form.
ii) Those procedures contained in the Solution Algorithm process, mainly in what regards the way it requires from a phenomenon the computation and assembling of the small matrices and vectors at the finite element level. Therefore, the Phenomenon Pattern presents an interface, which defines the ways it can be used by any solution algorithm during a simulation.

Particularly, the Phenomenon pattern represents an abstraction of the collection of commonalities found in the various ways the processes related to a phenomenon can be used by a solution algorithm in the context of the FEM. A further objective with such an abstraction is to make the representation of data sharing and dependence between different phenomena more intuitive and easier.

Before we go any further with our developments, it is important to acknowledge that the computation of matrices and vectors depends on a number of choices like, for instance, distribution of the order of approximation of the vectors fields over the mesh, definition of basis for the discrete spaces, numerical integration schemes, etc. Although the dealing with all those pieces of data are considered within the Phenomenon Pattern, we will not focus on this aspect, since it is not the more troublesome one. Instead, we will stress the relationship established by Phenomenon objects with each other, during the computation of coupled quantities, and with the solution algorithm (the software environment outside the set of Phenomenon objects).

This work presents some results obtained by researchers of the PLEXUS Project. The main objective of the PLEXUS project is the development of simulators for multi-physics and multi-scale systems with strong emphasis in reusability, maintainability and adaptability. The general set of methods currently being considered is the one known by the general name of the finite element method. However, strong attempts are being pursued in the sense of designing architectures, which could be used for other methods as well - for instance, finite volume method [4].

II. Context

In this section we explain the difficulties in dealing with coupled problems and devise some ways of circumventing them. The methodology of the explanation is to simplify the finite element method in such a way that the aspects, which produce difficulties, are retained, but the details, which are not relevant, are filtered out. In that direction, we start with the description of the levels of procedures, which can be identified during the development of simulators using the FEM. They are:
i. The finite element level:
- Sub-level of the production of matrices and vectors
- Sub-level of the error estimation
- Sub-level of the post-processing

ii. The solution level, composed of:
- Sub-level of the assembling of algebraic systems and its solution;
- Sub-level of interactions, which articulate solutions of different algebraic systems;
- Sub-level of loops and interactions involving progression in time and adaptation of models and discretizations.

The definition of those levels is important in the sense of software modularisation. But it does not indicate neither how entities belonging to different levels interact nor what data they share or depend upon. That is certainly very important for the definition of abstractions, which could standardize the way those entities behave and interact. In what concerns uncoupled problems, there are available finite element libraries, which are able of providing sufficient computational representation (abstraction) power in order to support the building of simulators in a reasonable time and with a high degree of reusability. Unfortunately, that is not true for coupled phenomena. Whenever a phenomenon depends on data from another phenomenon, that representation power breaks down. This is so because those libraries do not provide abstractions neither for the interaction between two phenomena nor for the interaction between a phenomenon and a solution algorithm.

Actually, coupled multi-physics problems make things very complex at both levels, that is, the finite element level and the solution level. The abstractions used for uncoupled systems can not be used efficiently - in the sense of reusability, adaptability and maintainability - in coupled systems. Those abstractions are not fit to adequately represent the data transfer and dependency, which may occur in coupled multi-physics problems, for one reason: there is a strong relationship between decisions at one level (the solution level) and computations at the other level (the finite element level).

The reason why this difficulty arises can be further clarified by analyzing the procedures at the lowest level, that is, the finite element level. Those procedures are related to the production (at each finite element) and assembling of the corresponding matrices and vectors, for each phenomenon. The small elemental matrices and vectors may be coupled with other phenomena, meaning that the computations of those quantities need pieces of information from other phenomena. Those pieces of information are defined at the solution level. Therefore, changing solution algorithms may produce widely spread changes across both levels.

In order to provide a further insight, consider that a phenomenon P_i is able of computing a set of quantities $\{Q_j(P_i)\}_{j=1}^{n_j}$. During the simulation each phenomenon has a fixed number of predefined states, which are represented by certain pieces of data computed during the many stages of a simulation. Assume that a given quantity $Q_j(P_i)$, for some j, is coupled to another phenomenon P_k. The definition of a coupling means that a certain number of states of P_k are used in the computation of $Q_j(P_i)$. Assume that P_k has the set $\{St_s(P_k)\}_{s=1}^{m_k}$ as the set of its states. P_i does not know a priori, which states of P_k are to be used in the computation of $Q_j(P_i)$, until the solution algorithm determines it (at the moment it requires P_i to compute $Q_j(P_i)$). This is a choice, which may be changed whenever a different solution algorithm is used. Suppose that only one state is required for the computation of $Q_j(P_i)$, that is, the state $St_r(P_k)$, for some r. This information is passed to P_i along with the requirement to compute $Q_j(P_i)$ and a data structure where it should assemble it. P_i should retrieve (or be given) the state $St_r(P_k)$ and then it will able of computing $Q_j(P_i)$ (on each finite element from the mesh) and assembling it into the given data structure. Notice that the decision about the state to be used in the computation of a coupled quantity is made at a substantially higher level than the level of the finite element. Furthermore, that state should be retrieved in order to be used by P_i. Those two aspects produce the major difficulties, which are present in the development of simulators for coupled phenomena. However, there are still other requirements, which make the problem even more intractable: (i) couplings may occur only on a part of the geometric domain (for instance, on a part of the boundary) or it may occur dynamically, as in contact problems; (ii) two phenomena, which are coupled in a geometric component may not share the geometric mesh, meaning that the use of a coupled state by one phenomenon needs the transfer of that state from the mesh of the other pheonomenon to the mesh of the former.

The described difficulties generate a problem that can be phrased as: What abstraction can adequately represent and encapsulate the information, relationships and processes pertaining to the finite element level and which are concerned to the numerical modeling of a natural phenomenon, in order to describe and implement phenomena abstractions in the context of coupled multi-physics systems? The next sections present a solution to this problem.

III. The Phenomenon Pattern

For the purpose of simplicity of explanation, a phenomenon is considered as a machine for computing small matrices and vectors and assembling them into given large matrices and vectors. The computation of those small matrices and vectors is performed on each finite element of a mesh. The following requirements have to be satisfied:

i) The computation of small matrices and vectors may be related to any geometric component among the set of geometric components, which define the physical domain of a phenomenon (a part of the boundary, for example).

ii) Each phenomenon should have a list of quantities, which it is able of computing.

iii) The definition of the coupled states needed for the computation of a certain quantity of a phenomenon should be defined only at the solution algorithm level. A phenomena should not be aware of the way it is used during a particular simulation.

iv) Two phenomena may share meshes on any common geometric part.

v) A phenomenon is coupled to vector fields (which happens to be from other phenomenon), which have a fixed definition (dimension, for instance). It is not a priori coupled to a specified phenomenon. The definition of the couplings between phenomena is delayed until the definition of a problem data for the simulation.

The developed solution, which should satisfy the above requirements, is the Phenomenon Pattern (called **Phenomenon** from now on). It can be viewed as a container with two acyclic graphs: the **PhenGraph** and the **GeomGraph** (see Figs. (1), (2), (3) and (4)). Both of them has exactly the same structure as graphs, but the pieces of data stored in each **GraphNode** are different from one graph to the other. We call by **PhenNode** and **GeomNode** the **GraphNode**'s from the **PhenGraph** and **GeomGraph**, respectively. In **GeomGraph** the geometry data is stored in a brep structure (boundary representation). That is, from the root of **GeomGraph** to its leaves one goes from the geometric entities of higher dimension (volumes, for instance) to the lower dimension ones (points). On the other hand, in **PhenGraph** one finds in each **PhenNode** a set of procedures, which are to be computed on the geometric entity of the respective **GeomNode**. Since geometry should be shared with other phenomena, it is important to have such a separation between them. Each **PhenNode** knows (has a reference to) its respective **GeomNode** and thus has access to its data, but the converse is not true.

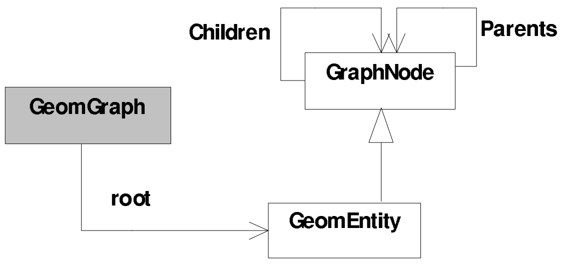

Fig. 1. Geometry Graph

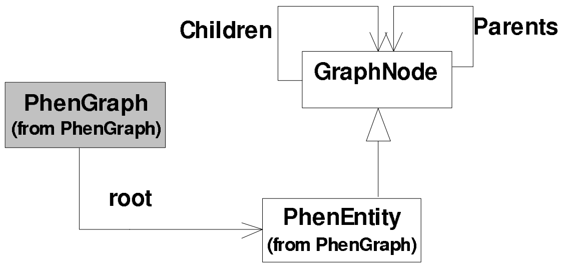

Fig. 2. Phenomenon Graph

Phenomenon also contains an indexed set of quantities $\mathbf{Q} = \{Q_i\}_{i=1}^{n}$ (see QTable in Fig. (4)), which represents the set of all quantities (small vectors and matrices) that it is able of computing and assembling into given structures (large matrices and vectors). Here n is the total number of quantities the current **Phenomenon** can compute and assemble. The procedures responsible for the actual computation of those quantities should stored in the **PhenNode**'s. This further implies the need for each **PhenNode**, say the j^{th} one, to have an indexed set $\mathbf{q}_j = \{Q_{j_k}\}_{k=1}^{n_j} \subset \mathbf{Q}$ (see qTable in Fig. (4), which represents all quantities the **PhenNode** j can compute.

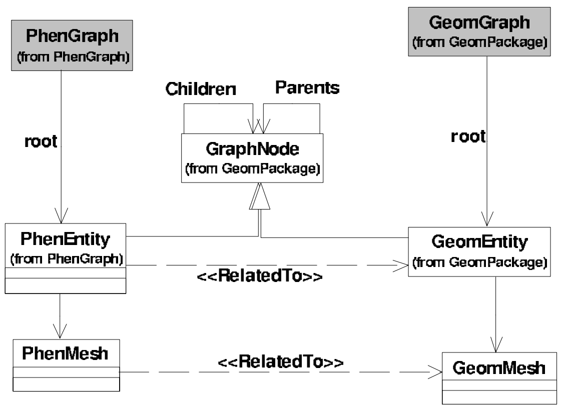
PhenGraph (from PhenGraph)
GeomGraph (from GeomPackage)
Children
Parents
GraphNode (from GeomPackage)
root
root
PhenEntity (from PhenGraph)
GeomEntity (from GeomPackage)
<<RelatedTo>>
PhenMesh
<<RelatedTo>>
GeomMesh

Fig. 3. Phenomenon-Geometry relationship

Each **PhenNode** needs some pieces of data in order to be able of computing a certain quantity. Those pieces of data are the following:

a) General Data (it serves all quantities):

- **GeomMesh**: it is the mesh of a the geometric entity stored in a **GeomNode**. It is a data structure, which contains all geometric finite elements (for instance, triangles, quadrilaterals, tretrahedra, hexagons, etc; it depends on geometric dimension and mesh generation methods used to obtain it). It belongs to its respective **GeomNode**.

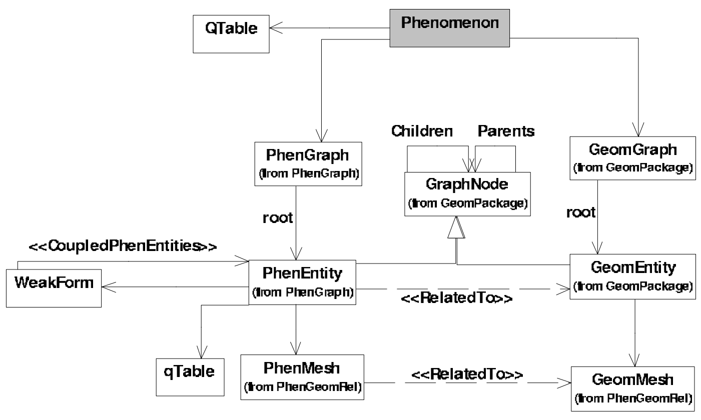
QTable
Phenomenon
Children
Parents
PhenGraph (from PhenGraph)
GraphNode (from GeomPackage)
GeomGraph (from GeomPackage)
root
root
<<CoupledPhenEntities>>
WeakForm
PhenEntity (from PhenGraph)
<<RelatedTo>>
GeomEntity (from GeomPackage)
qTable
PhenMesh (from PhenGeomRel)
<<RelatedTo>>
GeomMesh (from PhenGeomRel)

Fig. 4. Phenomenon Diagram

- **PhenMesh**: its is the mesh of the phenomenon. It is a data structure, which contains information about the approximation of the phenomenon's vector field on each geometric finite element (for instance, the order of a polynomial approximation). It is strongly related to the **GeomMesh**.

b) Specific Data (it is specific to a certain quantity) for a quantity Q_{j_k}, $k = 1, \ldots, n_j$:

- **CoupledPhenNodes**$_{j_k} = \{C_{j_{k_s}}\}_{s=1}^{n_{j_k}}$: it is the set of **GraphNode**s pertaining to other phenomena. They are used in order to obtain data from other phenomena, which are important in the computation of Q_{j_k}.

- **CoupledStates**$_{j_{k_s}} = \{S_{j_{k_{s_r}}}\}_{r=1}^{n_{j_{k_s}}}$: it is the set of states, which should be retrieved from the **CoupledGraphNode** $C_{j_{k_s}}$. Those states usually represent discrete vector fields.

Obs: It should be noted that the use (by a certain phenomenon) of a vector field from other phenomenon raises a number of issues. The main issue is the possibility that both **GeomMeshes** may be different. Due to lack of space we do not consider this important problem in detail in this work. What we should say is that we usually build **Phenomenon** derived classes, which are specially tailored for the task of transferring data from one mesh to the other. Such a **Phenomenon** has the following characteristics:

- It shares the **GeomGraph** with the coupler **Phenomenon**.
- The root of its **PhenGraph** has only the quantities needed for the transfer of data.
- Each quantity cited above is coupled to the respective **GraphNode** from the **PhenGraph** of the coupled phenomenon.
- Each quantity to be computed need only one coupled state - the state to be transfered.
- The **PhenMesh** from the coupled phenomenon is also needed, in order to obtain values of the coupled vector field in any desired point.
- Finally, there is a need for a search structure in order to find the finite element in the mesh of the coupled phenomenon, which contains a given point. This is important in order to perform integration procedures in one mesh using data from the other.

■

Now we come to the problem where data needed by procedures in a lower level (computation of quantities in each finite element) is defined at a higher level (the owner of **Phenomenon** objects). Before going directly to this point, it is better to explain how objects **Phenomenon** are used.

Phenomenon classes are supposed to be strongly reusable. They are very detailed pieces of software and contains information, which can be used in many different contexts and geometries. In order to achieve that state we had to make allow for the definition about the coupled states to be done dynamically at run time. That definition depends on the solution algorithms used in the simulation. The simulator is here considered as a pattern [3], [5], [10], which is - simply speaking - a workflow in the form of a tree and divided into four layers (see levels of procedures defined in Section(II)):

- **Kernel**: set of procedures related to algorithmic structures for the control of loops and interactions involving progression in time and adaptation models and discretizations;
- **Block**: set of procedures related to the articulation of solutions of different algebraic systems;
- **Group**: set of procedures related to the assembling and solution of algebraic systems, together with the execution of diverse operations with matrices and vectors.
- **Phenomenon**: encapsulates the set of procedures related to the production of small matrices and vectors and to their assembling in given larger data structures. It also performs other computations related to post-processing and error estimation.

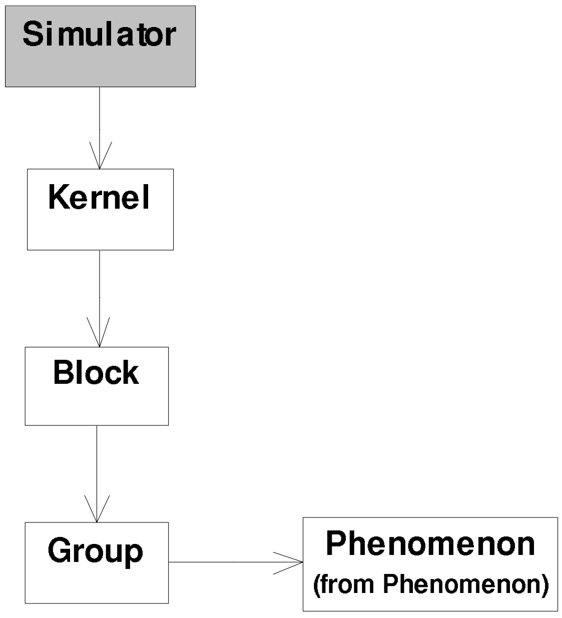

Fig. 5. Simulator Diagram

The simulation starts with the execution of the root of the **Kernel**, which uses services provided by a set of **Block**s, which in turn uses services from a set of **Group**s. Each **Group** owns a set of **Phenomenon** objects, which are used to perform the production small matrices and vectors, assembling them into given (by its **Group**) larger data structures. The intention of this work is not going too far in the explanation of the details of the interactions between objects across the levels of the **Simulator** Pattern (see Fig. (5)). Before we proceed further with the description of the lowest level - represented by the **Phenomenon** Pattern - we need some pieces of information about the structure of **Group** classes (see Fig. (6)):

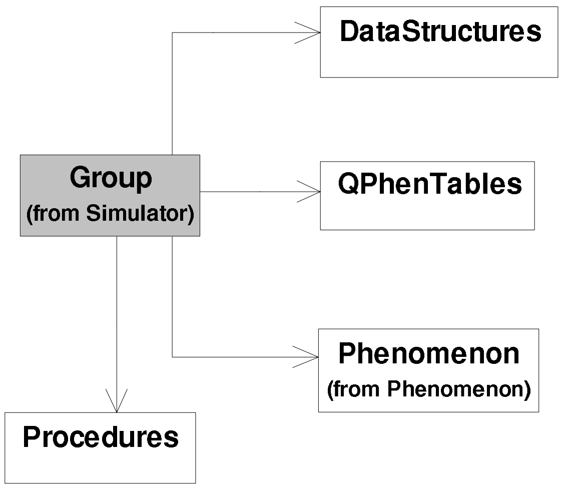

Fig. 6. Group Diagram

- It has a set of indexed data structures (see DataStructures in Fig. (6)), which can be shared upon demand by its own **Phenomenon** objects.
- It has a set of tables, which can be dynamically programmed in order to contain information about the computation of matrices and vectors, which **Phenomenon** objects contribute to them and how they should do it (that is, which **Phenomenon** quantities are to be used and, for each one of them, what are their coupled states)

Therefore the **Group**s level is the level where definitions about the coupled states take place and are conveyed to each **Phenomenon** object during a demand for the computation of a certain quantity. The coding of the software components for a **Group** class needs previous knowledge about the set of indexed data structures from all other **Group** classes. Because of that feature, a **Group** class is much less reusable than the classes from the other levels. Whenever a solution algorithm is changed, many **Group** classes may need to be eliminated from the simulators. However, all **Phenomenon** classes should have been implemented in such a way that no modification at all will take place. Some minor reprogramming of **Block** and **Kernel** procedures, but their coding as a set of procedures articulated as workflows will accommodate that without much work.

Now we are in a position to describe in a more detailed fashion the usage of a **Phenomenon** (call it **P**) object by its owner, that is, a **Group** object (call it **G**):

- **G** is asked (by its **Block**) to compute a certain quantity Q_g.
- **G** retrieves from one of its tables the references to all **Phenomenon** objects, which contribute to Q_g. **P** is among them.

- **G** retrieves from another table a set of data for each contributing **Phenomenon** object. This set of data for the ith-object contains information such as: (I) a set $\mathbf{qt}_i = \{q_{i_j}\}_{j=1}^{n_i}$, containing all quantities from the ith-Phenomenon, which contributes to form Q_g; (II) for each q_{i_j}, a set $\mathbf{CS}_{i_j} = \{S_{{i_j}_k}\}_{k=1}^{n_{i_j}}$, containing the coupled states to be used in its computation - the order in which those states are given is important, since it they may be from different phenomena; (III) a reference to a data structure (say K_g) where the quantity should be assembled.
- Suppose, now, that **P** is given a demand to compute a certain quantity Q_r, together with a set of indexes for the coupled states, say $\mathbf{CS}_{p_r} = \{S_{{p_r}_k}\}_{k=1}^{n_{p_r}}$ and a reference to K_g.
- Then, **P** transfer the demand to its **PhenGraph**, which in turn send it to its root (say **R**).
- **R** checks if it is able of computing the desired quantity. If so, it sends the demand to one of its objects (say **W**), which is responsible for the computation and assembling of Q_r. **R** sends, together with the data already defined, a set of references, say **CG**, to the respective **GraphNode**s from the **PhenGraph**s of the coupled **Phenomenon** objects.
- **W** retrieves the set of coupled **PhenMesh**s (say, **CPM**) and the respective set of discrete vector fields (say **VF**) from the respective set of coupled **GraphNode**s. This is done by sending a request for the states contained in CS_{p_r} to the respective **GraphNode**s contained in **CG**. Each coupled **GraphNode** from **CG** is able of asking its owner **Group** for the desired state and sending it back to **W**.
- **W** traverse the **PhenMesh** (given by **R**) together with the coupled ones (contained in **CPM**) and is able of computing, for each finite element, the quantity Q_r - using the coupled discrete vector fields (contained in **VF**) - and of assembling them into K_g.
- If **R** is not able of computing the desired quantity, or if it has already computed it, it will transfer the demand recursively to its children **GraphNode**s together with the same data it received. The process goes on recursively, until there are no children **GraphNode**s to be reached.

Obs: Note that, in the description of the computation of Q_r vy **R**, we have considered that all **PhenMesh** objects (that given by **R** and those contained in **CPM**) share the same **GeomMesh** object. Therefore they could be traversed at the same time. ■

IV. Conclusions

We have presented a pattern called **Phenomenon** in order to cope with the difficulties found in the development of simulators for coupled multi-physics problems. It was identified that the solution algorithms employed by the simulators provide data at a higher level, which determines many procedures at a lower level. Thus, small changes in the solution algorithm could generate the need for widespread changes along the simulator software, implying in strong reprogramming. A separation of concerns was important to identify that the simulator could be divided into four well defined hierarchical levels. The part of a solution algorithm where pieces of data are generated to be used at a lower level was concentrated at a level called the **Group** level. The part at a lower level where those pieces of data are used was identified as the **Phenomenon** level. An abstraction for the computational representation of the **Phenomenon** level was described as a pattern. It was shown that his pattern (**Phenomenon** Pattern) represents adequately all data pertaining to the discrete behavior laws of a phenomenon (production of small matrices and vectors), together with the needed sharing and transfer of data between **Phenomenon** objects and between **Phenomenon** objects and the solution algorithms. An important consequence was that all information and procedures, which are very specific of a solution algorithm (less reusable) are encapsulated in the **Group** level, while all information and procedures, which are strongly are encapsulated in the **Phenomenon** level. Experiences with prototypes have shown a tremendous improvement in: (i) the time spent in the development of simulators; (ii) the reusability of software components, opening the way for the building of repositories; (iii) the correctness of software components. References of previous and related work can be found in [5]-[10]

References

[1] Committee on Theoretical and Applied Mechanics, Research Directions in Computational Mechanics. A report of the United States Association for Computational Mechanics, 2000. Available at http://www.usacm.org/org_cm.htm, accessed on 10/01/2004.

[2] Viceconti, M., Replication of Numerical Studies. Personal Communication, posted at biomch-l@nic.surfnet.nl. Sent on: March 27, 2002 3:03 pm Subject: BIOMCH-L: BioNet controversial topic no. 5.

[3] Santos, F.C.G., Lencastre, M., Vieira, M. GIG-Pattern. The Third Latin American Conference on Pattern Languages of Programming, Pernambuco, Brazil, 2003.

[4] Lencastre, M., "Conceptualisation of an Environment for the Development of FEM Simulators", Ph.D. thesis, Centro de Informtica, Universidade Federal de Pernambuco, Brazil, 2004.

[5] Lencastre, M.; Santos, F., "An Approach for FEM Simulator Development"; Journal of Computational Computational and Applied Mathematics, Vol. 185, issue 2, 2006.

[6] Lencastre, M.; Santos, F.; Arajo, J., "A Process Model for FEM Simulation Support Development. Proceedings of the Summer Computer Simulation Conference (SCSC 02), San Diego, California, 2002.

[7] Lencastre, M.; Santos, F.; Rodrigues, I., "FEM Simulator based on Skeletons for Coupled Phenomena". Proceedings of the 2nd Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP'2002 Conference), pp.35-48, Brazil, 2002.

[8] Lencastre, M.; Santos, F.; Rodrigues I., "FEM Simulation Environment for Coupled Multi-physics Phenomena", - Simulation and Planning In High Autonomy Systems, AIS2002; Theme: Toward Component-Based Modeling and Simulation. pp. 259-266. A publication of the Society for Modeling and Simulation International, Portugal, 2002.

[9] Lencastre, M.; Santos F.; Vieira, M., "Workflow for Simulators based on Finite Element Method". Proceedings of the International Conference on Computational Science (ICCS 2003), Melbourne, Australia and Saint Petersburg, Russia. 1(2), pp. 555-565, Springer Verlag, 2003.

[10] Lencastre, M.; Santos, F.; Vieira, M., "GIG-Pattern". Proceedings of the 3rd Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP'2003), pp. 293-307, Pernambuco, Brazil, 2003.