Automatic Development of High Performance Multi-Physics Simulators

Félix Christian Guimarães Santos, Email: flxcgs@yahoo.com.br
Eduardo Roberto R. de Brito Junior, Email: errbj@yahoo.com.br
José Maria Bezerra, Email: zemaria@ufpe.br
Federal University of Pernambuco
Department of Mechanical Engineering
Rua Acadêmico Hélio Ramos, s/n - Recife - PE 50740-530 - Brazil

Abstract—**MPhyScas - Multi-Physics and Multi-Scale Solver Environment - is a computational system aimed at supporting the automatic development of simulators for coupled problems, developed at the Department of Mechanical Engineering of the Federal University of Pernambuco - Brazil. It provides a framework, which is flexible enough to accommodate representations for all levels of computation that can be found in simulators based on the finite element method. MPhyScas is built on a set of a powerful language of patterns supporting abstractions for solution algorithms; phenomena, geometric entities; phenomenon-phenomenon and phenomenon-geometry relationships and others, together with a library of low level entities - like finite elements, reference finite elements, numerical integration tools, and so on. In despite of its completeness in what regards all stages of a multi-physics simulation, the current version of MPhyScas produces sequential simulators only. Thus, it does not support any kind of communication between its computational entities besides those defined by direct references (pointers). In this work we present the architecture of an improvement of MPhyScas, called MPhyScas-P (MPhyScas Parallel), which can be used for the automatic development of either sequential or parallel simulators. We take an advantage of the architecture in layers of MPhyScas in order to define a hierarchical parallel computational scheme in such a way that communication procedures are automatically identified, localized and built. That hierarchy also provides a natural way of defining data structures and access dynamics for all memory levels, providing simpler ways of dealing with non-uniform memory access patterns. Some preliminary results obtained with a prototype will be shown and analyzed.**

Index Terms—**Finite element method, Simulator, Multi-physics, Coupled phenomena**

I. Introduction

MPhyScas (Multi-Physics Multi-Scale Solver Environment) is an environment dedicated to the automatic development of simulators based on the finite element method. The term multi-physics can be defined as a qualifier for a set of interacting phenomena defined in space and time. These phenomena are usually of different nature (deformation of solids, heat transfer, electromagnetic fields, etc.) and may be defined in different scales of behavior (macro and micro mechanical behavior of materials). A multi-physics system is also called a system of coupled phenomena. If two phenomena are coupled, it means that a part of one phenomenon's data depends on information from other phenomenon. Such a dependence may occur in any geometric part, where both phenomena are defined. Other type of data dependence is the case where two or more phenomena are defined on the same geometric component and share the geometric mesh. Multi-physics and multi-scale problems are difficult to simulate and the building of simulators for them tend to be very costly in terms of time spent in the programming and testing of the code. The main reason for that is the lack of reusability. A detailed discussion can be found in [1]-[2].

Usually, simulators based on the finite element method can be cast in an architecture of layers. In the top layer global iterative loops (for time stepping, model adaptation and articulation of several blocks of solution algorithms) can be found. This corresponds to the overall scenery of the simulation. The second layer contains what is called the solution algorithms. Each solution algorithm dictates the way linear systems are built and solved. It also defines the type of all operations involving matrices, vectors and scalars, and the moment when they have to be performed. The third layer contains the solvers for linear systems and all the machinery for operating with matrices and vectors. This layer is the place where all global matrices, vectors and scalars are located. It is also responsible for the definition of the finer details for the assembling of matrices and vectors. The last layer is the phenomenon layer, which is responsible for computing local matrices and vectors at the finite element level and assembling them into global data structures.

The definition of those layers is important in the sense of software modularization. But it does not indicate neither how entities belonging to different layers 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 layers behave and interact. The architecture of MPhyScas presents a language of patterns in order to define and represent not only a set of entities in each layer - providing the needed layer functionalities - but also the transfer of data and services between the layers. Thus, MPhyScas is a framework that binds together a number of computational entities, which were defined based on that language of patterns, forming a simulator. Such a simulator can easily be reconfigured in order to change solution methods or other types of behavior [3]-[4]. Almost every single piece of code that constitute MPhyScas computational entities in a simulator can be reused in the building of other different simulators. This makes the simulators produced by MPhyScas strongly flexible, adaptable and maintainable.

However, the original architecture of MPhyScas provides

Proceedings of the 2008 High Performance
Computing & Simulation Conference ©ECMS
Waleed W. Smari (Ed.)
ISBN: 978-0-9553018-7-2 / ISBN: 978-0-9553018-6-5 (CD)

support for the automatic building of sequential simulators only. For instance, it does not have abstractions that could automatically define the distribution of data and procedures and their relationships across a cluster of PC's. In this work we briefly present MPhyScas architecture for sequential simulators (called MPhyScas-S from now on) in order to proceed with the main part of the work, i.e., the definition of a new parallel architecture. This new architecture, called MPhyScas-P, should satisfy a number of new requirements, including the support for the parallel execution of the simulators in clusters of PC's. MPhyScas-S is a framework with the support of an extended finite element library and a knowledge management system. MPhyScas-P uses the same extended library and knowledge management system from MPhyScas-S (with minor differences). It also makes use of the concept of layers already used in MPhyScas-S in order to define a hierarchical parallel structure. Such a hierarchy is useful for the automatic definition of synchronization schemes; data partition and distribution procedures; inter-process communication patterns and data management across several levels of memory. Procedures are automatically specialized for the pre-processing; the simulation and the post-processing phases, depending on the hierarchical distribution and on the characteristics of the hardware being used. Also, only two types of communication between processes during a simulation, and patterns are defined for their representation.

This work is organized as follows: first the architecture of MPhyScas-S is briefly described paving the ground for describing MPhyScas-P's architecture. Next, comments on some relevant related work are provided in order to build a context for this article. After that is done, the architecture of MPhyScas-P is presented, followed by a description of its behavior (in the context of the work load and flow requirements). In the end some conclusions are drawn. Our purpose here is more descriptive than analytic. However, whenever needed some comments will be provided in order to make things a bit clearer. In the end some conclusions are drawn.

II. The Architecture of MPhyScas-S

The architecture of MPhyScas-S [5], [4] establishes a computational representation for the computational layers using some design patterns (see Figure 1), where the **Kernel** Level represents the global scenery level, the level of the solution algorithms is represented by the **Block** Level, the level of solvers is represented by the **Group** Level and the phenomena level is represented by the **Phenomenon** Level. The definition of this structure is aimed at improving the quality of simulators design. The defined architecture attempts to fill in the existing gap in the development of FEM simulators for multi-physics and multi-scales problems. The main requirements of this architecture are: (i) Flexibility in the development of simulators; (ii) Extensibility of simulators through the integration of components and (iii) Improved reusability of code, data and models.

The architecture of MPhyScas-S is shown in Figure 2. The Static Library allows the maintenance of data employed in the building of simulators and simulations. Those data includes: methods (mesh generation, numerical integration, for instance), functions (constitutive parameters, for instance), algorithms and phenomena. The Pre-Processor produces **Data for Simulation** and builds the **Simulator** using the **Static Library**. The **Data for Simulation** represents the input data in a simulation as it is used by the **Simulator**. The **Simulator** is responsible for the execution of a simulation. The **Simulator** uses the **Data for Simulation** and produces the **Results of the Simulation**. The **Viewer** uses the **Results of the Simulation** and the **Data for Simulation** to produce the visualization of the simulation results.

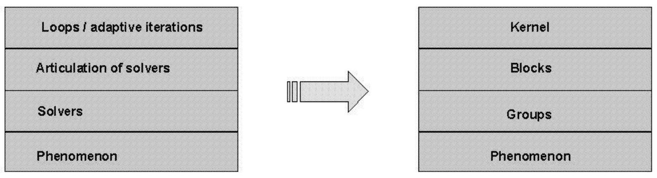

Fig. 1. Computational representation for the layers of the simulator

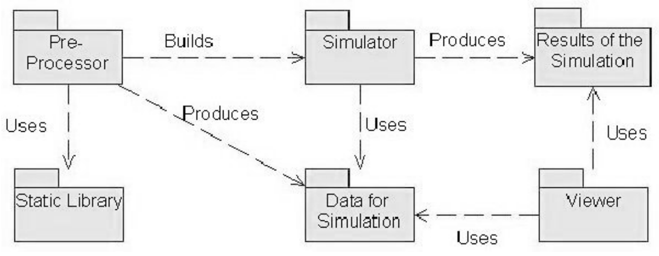

Fig. 2. Architecture of the MPhyScas

The **Simulator** is considered as a pattern [6], [3] and [7] (see Figure 3), which, simply speaking, is a workflow in the form of a DAG (Direct Acyclic Graph) and divided into four layers, which strongly follows the slices of concerns for each layer, already cited in the introduction:

i) **Kernel**: it is responsible for initialization procedures (transferred to **Blocks** in the lower level); for global time loops and iterations; for global adaptive iterations and for articulation of activities to be executed by the **Block**s in the lower level. The **Kernel** stores system data related to the parameters for its loops and iterations;
ii) **Block**: it is responsible for the transfer of incoming demands from the **Kernel** to its **Group**s in the lower level (initialization procedures, for instance); for **Block** local time loops and iterations (inner loops and iterations inside a global time step, restricted to groups of phenomena); for procedures inside time stepping schemes; for **Block** local iterations (restricted to some groups of phenomena, like in a Newton-Raphson iteration, for instance); for **Block** local adaptive iterations (restricted to some groups of phenomena); for operations with global quantities (transferred to **Group**s in the lower level, which are the owners of global quantities). The **Block**s serve the **Kernel** level. Each **Block** is responsible for a certain number of **Group**s, which can not be owned by other **Block**. All demands from a **Block** to the lower level should be addressed to its **Group**s. The **Block**s store system data related to parameters for their own loops and iterations

and parameters for their procedures;

iii) **Group**: it is responsible for the transfer of incoming demands from its **Block** to **Phenomena** in the lower level (initialization procedures, for instance); for the assembling coordination and solution of systems of linear algebraic equations (the method used depends on the solver component); for operations with global quantities (by demand from its **Block**), for articulation of activities to be executed by its **Phenomena** in the lower level (basically concerned with computation and assembling of global matrices and vectors). The **Group**s serve their respective **Block**s. Each **Group** is responsible for a certain number of **Phenomena**, which can not be owned by other **Group**. All demands from a **Group** to the lower level should be addressed to its **Phenomena** only. The **Group**s store global matrices, vectors and scalars and store the **GroupTask**s, which are objects encapsulating standard procedures, where articulation of the **Group**'s **Phenomena** are needed. The **GroupTask**s are programmable and their data are standard pieces of information, depending only on the type of the **GroupTask**.

iv) **Phenomenon**: it is responsible for the computation of local matrices, vectors and scalars (**Phenomenon** quantities); for operations involving matrices and vectors at the finite element level and their assembling into given global matrices and vectors. The **Phenomena** serve their respective **Group**s. The **Phenomena** store data related to constitutive parameters or other parameters, which are specific of the respective Phenomenon; store the geometry where the respective **Phenomenon** is defined (different **Phenomena** may share a geometry or a part of it); store **WeakForm**s, which are tools for computing and assembling quantities defined on a certain part of the geometry. A **WeakForm** may be active or not. Only active **WeakForm**s can be used during a simulation. A **WeakForm** may store parameters, which are related to specific simulation data (for instance, functions for the definition of boundary conditions or parameters needed for the computation of a quantity, which should be given together within a simulation data set). The **Phenomenon** should store methods, which are tools to be used in certain **Phenomenon** specific tasks. For instance, those tasks can be generation of geometric and **Phenomenon** meshes, numerical integration at the element level, shape functions, etc.

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 of local matrices and vectors and the assembling of them into given (by the **Group**) global matrices and vectors.

The states that define the configuration of each **Phenomenon** object are stored in the respective **Group** object, where solvers are located. This is convenient due to the fact that the **Group**'s layer is responsible not only to assemble and solve algebraic systems, but also to operate with scalars, vectors and matrices in response to requests from a **Block** (see Figure 5).

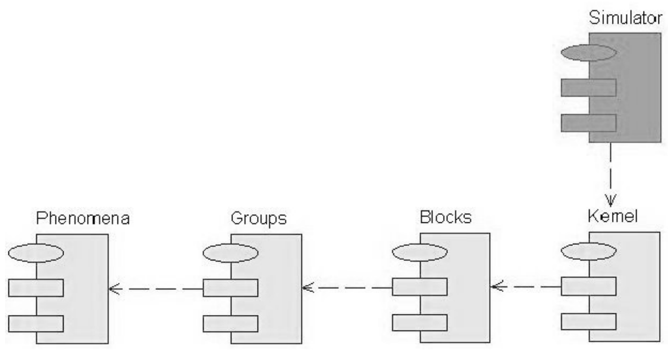

Fig. 3. Simulator diagram

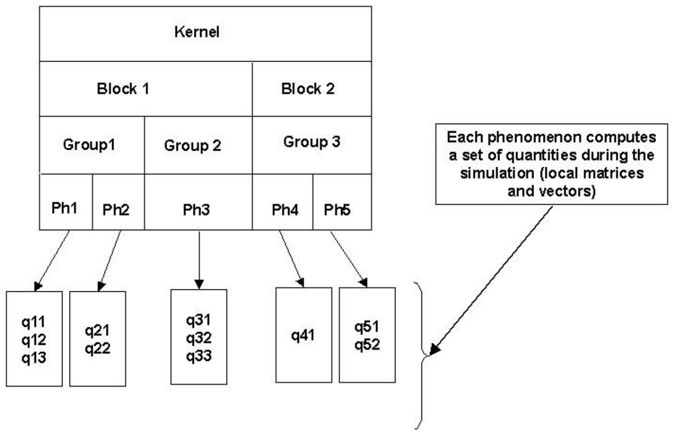
Kernel
Block 1
Block 2
Group1
Group 2
Group 3
Ph1
Ph2
Ph3
Ph4
Ph5
Each phenomenon computes a set of quantities during the simulation (local matrices and vectors)
q11 q12 q13
q21 q22
q31 q32 q33
q41
q51 q52

Fig. 4. Each Phenomenon object is able of computing a set of quantities during a simulation

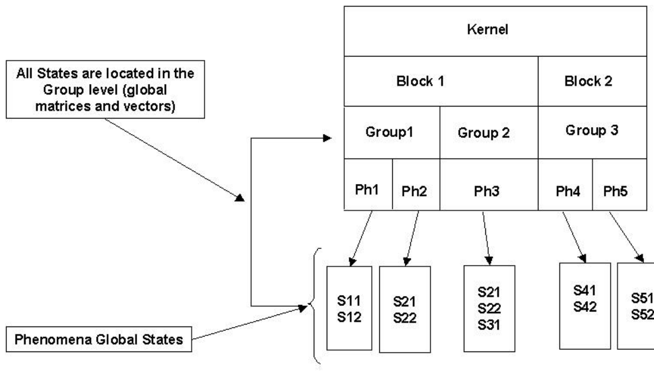

Fig. 5. Each Phenomenon object has its own set of states, which is stored in its Group object

A quantity that a **Phenomenon** object can compute and assemble may be coupled to other **Phenomenon**'s states (one or more) as it is depicted in Figure 6. MPhyScas-S provides all the machinery to make this procedure automatized following the specification of some data related to the place where coupling occur; handlers for the states and a reference to the coupled **Phenomenon** object, which should be given to the object responsible for the computation.

For further detailed information on MPhyScas-S see [4]. In the next section we provide some relevant related work.

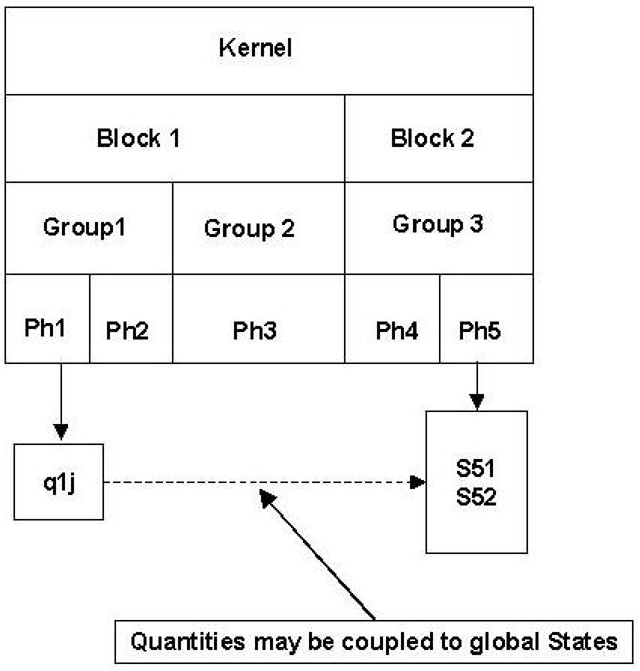

Fig. 6. A quantity computed by a Phenomenon object may be coupled to a state from other Phenomenon object

III. Related Work

Definition and building of computational frameworks that support programming of simulator for multiphysics problems has been a very active area in the last decade. For the sake of providing a simpler context for the present work, we classify current research efforts into only two classes: (i) libraries, which, besides providing important abstractions supporting data, procedures and relationships for coupled physics simulation, do not provide a structural guidance (architecture) for the building of simulators, and (ii) frameworks, which provide a deep structure of abstractions and patterns in the form of an architecture. We will comment more on the second class, since MPhyScas is a framework with those characteristics. As examples of class (i) we cite the Component Template Library (CTL) [8] and Comsol [9]. CTL provide abstractions that support the building of solutions algorithms for loose and tight coupling (procedures in the level of Group and Phenomena). It is an implementation of the component concept with an RMI semantic similar to CORBA or Java-RMI components, which can be used to build complex parallel simulators. It is sophisticated in the sense that allows component in several languages (C, C++, FORTRAN) and different communication models (RMI, MPI, PVM, threads) among other features. Comsol is a commercial package and not much of its internal behavior is publicly exposed. Nevertheless, it provides abstractions that allow users to define coupled procedures through handlers to vector fields and provide encapsulated access to high performance computing. It has a sophisticated GUI with CAD and visualization modules. However, besides varying levels of support for HPC, neither one of them provides structural guidance for simulators (levels Kernel through Phenomena), leaving that task to the user.

In the class of frameworks (class (ii)), one can find powerful packages, such as Uintah [10], Cactus [11] and Sierra [12]. They are substantially different and have been applied to extremely sophisticated simulations. Since the architecture of MPhyScas is closer to Sierra's architecture, we will comment on this framework with more detail. Uintah Computational Framework and Cactus Framework consist of a set of software components and libraries that facilitate the solution of Partial Differential Equations (PDEs) on Structured AMR (SAMR) grids using hundreds to thousands of processors. Although they do not provide a structure for simulators as done by Sierra and MPhyScas, they are in this class due to how they bind components together; define and use coupling information and provide access to high performance (e.g. parallel) processing through a shared service infrastructure. That characterizes them as having a deep interoperability system. Uintah uses CCA (Common Component Architecture) [13] for designing and describing components interfaces. It does not have a pre-defined structure for simulators. Thus, in order to provide framework functionality it defines on top of its primary set of abstractions another set of components and supporting libraries, which targets the solution of PDE's on massively parallel architectures. This set is called Uintah Computational Framework (UCF). UCF builds a graph called **TaskGraph**, which describes the data dependencies between the various computation steps in a simulator. Also, it defines a simulator's workflow as a direct acyclic graph of **Task**s. Communication between Tasks is made through a DataWareHouse, which provides the illusion that all memory is global. If a Task correctly describes its data dependencies, then the data stored in the DataWareHouse will match the desired data (variable and region of state). Communication is scheduled by a local scheduling algorithm that approximates the true globally optimal communication schedule. Because of the flexibility of single-assignment semantics, the UCF is free to execute tasks close to data or move data to minimize future communication [10].

The following nice summary of Cactus structure is found in [14]: ”The Cactus **Flesh** acts as the coordinating glue between modules that enables composition of the modules into full applications. The Flesh is independent of all modules, includes a rule based scheduler, parameter file parser, build system, and at run time holds information about the grid variables, parameters, methods in the modules and acts as a service library for modules. Cactus modules are termed **Thorns** and can be written in Fortran 77 or 90, C or C++. Each thorn is a separate library providing a standardized interface to some functionality. Each thorn contains four configuration files that specify the interface between the thorn and the Flesh or other thorns (variables, parameters, methods, scheduling and configuration details). **Drivers** are a specific class of Cactus Thorns that implement the model for parallelism. Each solver thorn is written to an abstract model for parallelism, but the Driver supplies the concrete implementation for the parallelism" (see also [11]).

Sierra framework provides a structural guidance in layers composed of (from top to bottom) **Application**, **Procedure**, **Region** and **Mechanics**. Application articulates user-provided algorithms in order to establish high-level activities. It uses services from a set of Procedures, which can freely articulate Regions using a set of user-provided algorithms. A Region defines activities, which are related to a fixed geometric region, for which a mesh is provided. Those activities are defined by user-provided algorithms. It uses services provided by a set of Mechanics. In order to perform the desired work, a Mechanics uses a set of MechanicsInstances and a set of user-provided algorithms. A Mechanics implements procedures related to a specific physics - defined on a subset of its Region's mesh - and its MechanicsInstances are responsible for atomistic operations defined on a subset of its Mechanics' mesh. A Mechanics may use another set of Mechanics, building more layers downwards. This may be used in multiscale computations, where a lower level Mechanics is used to compute constitutive data for a MechanicsInstance of its parent Mechanics [12]. If a Mechanics A needs data from another Mechanics B (provably defined in another Region), the Advanced Services of Sierra provide means to transfer mesh-dependent data from one mesh to the other. The result is then stored in the Region of Mechanics A. The SIERRA Framework Core Services manage the parallel distribution of mesh objects for an application. Management of a parallel distributed mesh is defined in three parts: (i) policies and distributed mesh sets, relations, and data structures; (ii) parallel operations that do not modify the distributed-mesh data structures, and (iii) operations that modify the distributed-mesh data structures. Sierra has a sophisticated management system for parallel operations, which is strongly supported by its defined topology. As far as the authors are concerned Sierra supports only SPMD (single process multiple data) type of parallel processing.

Clearly, it is possible to provide a parallel between the architecture of MPhyScas and that of Sierra. Application relates to Kernel; Procedure relates to Block; Region relates to Group; Mechanics relates to Phenomenon and MechanicsInstance relates to WeakForm (geometry-related atomistic piece of code). However, there are several and important notes that should lead to marked differences:

i) MPhyScas is strongly concern-oriented, instead of procedure-oriented or context-oriented. Concerns are more easily mapped into requirements and architectures. Also, concerns are related to the fundamentals of the classes of problems being tackled and are less vulnerable to programming traditions and limitations. Adequate separation of concerns may lead to more reusable, maintainable and adaptable code. Therefore, MPhyScas was built to satisfy a nested set of concerns (functional and non-functional) related to the numerical approximation to solutions of partial differential equations (mainly those solutions defined by the finite element method). For instance, layers in MPhyScas are slices of the code following a set of concerns already cited before. Sierra is more procedure-oriented.

ii) The specification for the Kernel (simulator's scenery) is more detailed than that for Sierra's Application. It has only one shallow algorithm limited by the designed responsibilities of the layer of Blocks. Of course, that algorithm can be freely designed, but should satisfy the concerns specified by what we call the Scenery of the Simulation (it can be understood as a general specification for the simulator, which gets more refined when requirements for the other layers are detailed).

iii) MPhyScas's Block is quite similar to Sierra's Procedure in their generality (their behavior is determined solely by their algorithms). However, one Block never shares a Group with other Block. This constraint does not apply to the relationship between Procedure and Region in Sierra framework. The justification for both Procedure and Block is the need for the articulation (in adaptive iterations, nonlinear solver iterations, and other situations) of sets of solvers involving different phenomena (physics). As Sierra does it, MPhyScas limited the depth of this layer in a slice of a generic simulator, where activities related to time stepping methods, nonlinear iterations and other processes are defined and articulated for one fixed set of subsets of phenomena . However, in MphyScas each Block also has the responsibility to define its **cone of influence** in a disjoint way. The reason is that MPhyScas would like to support dynamic exchange of components in all levels of computation. Therefore if two Blocks were allowed to establish relationships to the same Group, concerns related to both would get messed up, making it extremely difficult to define the parts of code affected by changes in one Block.

iv) A Region in Sierra is based on operations (Mechanics and algorithms) and vector fields, which are defined on a geometric entity and its geometric mesh. Transfer of vector fields from one mesh to the other is provided by the Advanced Services of Sierra. All vector fields in a Region are defined in subsets of the same mesh. The motivation for the Group in MPhyScas is based on a set of Phenomenon objects and their data, which participate in formation of linear monolithic algebraic systems. Thus, all data needed to assemble and solve those algebraic systems are stored in the Group. Thus, there might be Phenomenon objects in a Group defined on different meshes. That becomes manageable because a Group does not know anything neither about geometry domains nor about meshes. The transfer of vector fields between meshes is performed by a especially designed Phenomenon (instantiated and executed as a normal Phenomenon object). Those pieces of information are located in the respective Phenomenon objects. There are special data structures that allow two Phenomenon objects to share the mesh of a geometric entity, whenever they are defined in that geometric part [4]. All matrices and vectors are stored in the Group and their relationship with the Group's Phenomenon objects is described by user-provided data. The location of matrices and vectors in the Group was motivated by the location of linear solvers in the Group. All dependencies between Phenomenon objects are resolved in the Phenomenon layer. Besides the solvers, a Group is entirely programmable; does not depend on other user-defined algorithm and does not share Phenomenon objects with other Group (therefore providing the influence cone). There are other differences, but the cited references are able of providing further information.

v) Mechanics in Sierra encapsulate procedures related to a particular physics. It articulates its MechanicsInstances and algorithms in order to provide the computation of quantities and the assembling of them. MPhyScas provide those functionalities with Phenomenon objects and their activated WeakForm objects. A Phenomenon object accepts algorithms for activities such as numerical integration, error estimation, mesh adaptation (geometric and phenomenon meshes), shape functions (trial and test), mesh generation (geometric and phenomenon meshes). A Phenomenon has two types of meshes: geometric and phenomenon. Phenomenon meshes describe the distribution of polynomial order of approximation over the geometric mesh. It seems Sierra does not support p and h-p adaptivity, because it does not data structures for such procedures. This certainly would complicate transfer procedures and the way coupled vector fields are used in Sierra. That is supported by MPhyScas and is one of the concerns, which was considered when placing meshes at the Phenomenon layer.

Both architectures (MPhyScas' and Sierra's) were developed independently. The first definition of the MPhyScas' architecture was published in 2001. Nevertheless, they present a similar structure with some marked differences (mainly in the execution graph and placement of some data structures), which get sronger in MPhyScas-P, where the tree structure of the architecture of MPhyScas is used in the control of the simulator execution on a cluster of PC's (see next section). It is important to note that while MPhyScas-P is in the beginning of its development, Sierra is already a mature, complex, fully developed system with far more functionalities than MPhyScas-P. MPhyScas-P is being developed to be applied in a production environment (analysis of material degradation for the petroleum industry).

IV. The Architecture of MPhyScas-P

In modern clusters of PC's one can identify at least four hierarchical levels of different procedures and/or memory usage:

i) **Cluster Level**: it is composed of all processes running in all machines being used in a simulation.

ii) **Machine Level**: it is composed of all processes running in one individual machine among all those used in a simulation

iii) **Processor Level**: it is composed of all processes running in one individual processor among all those running in one individual machine.

iv) **Process Level**: it is composed of one single process running in one individual processor (provably multi-core) among all other processes in this same processor. It can be divided into two groups:

   iv.i) **Core Sub-level**: it is composed of all parts of the code from one individual process, which is not strongly hardware specific.

   iv.ii) **Software-Hardware (SH) Sub-level**: it is composed of all parts of the code from one individual process, which is strongly hardware specific (cache management, fpga acceleration, etc.)

Whenever the architecture of a computational system allows for a hierarchy of procedures, it may be a good idea to define a hierarchy of processes in such a way that few of them would accumulate some very light management tasks. The benefits for this strategy include: (i) procedures can be hierarchically synchronized (from coarse to fine grain), reducing management concerns and increasing correctness; (ii) since locality concerns change along the hierarchy levels, memory management can become more and more specialized from top to bottom. Communication processes can also benefit from locality knowledge; (iii) The hierarchy allows for the encapsulation of concerns, making it easier the design of exchangeable components. Besides the natural benefit of this aspect, it also allows for the adaptation of the code to new hardware and software technologies, without incurring in heavy reprogramming in all levels of the hierarchy.

A. Interprocess Communication Process

Next we provide a description of how interprocess communication is considered in MPhyScas-P architecture. Communication between processes is a very important issue for problems with only one physics and gets even more crucial for multi-physics problems. The cause for that is the fact that for those types of problems communication is not needed only to complete information during linear algebra operations, but also to transfer information (vector fields and meshes) from one phenomenon to the other. Furthermore, such a data dependence between phenomena does not occur only on the boundaries between two mesh components, but on any geometric entity. There are other complication factors that contribute to make things even worse. Changing solution algorithms means that the linear systems, which are going to be solved, may be dramatically changed. For instance, changing from a monolithic scheme to an operator splitting one will require the solutions of several different coupled linear systems (inside an iteration loop) instead of only one.

The architecture of MPhyScas-S has satisfactorily solved those problems related to data dependence and sharing between Phenomenon objects for the sequential processing. It is also able of representing solution algorithms in such a way that the entire simulator can be adapted with a minimum amount of reprogramming (maximum amount of reuse). However, the essential difference, when considering parallel processing, is the appearance of interprocess communication procedures, which can be of four types:

i) **Communication during linear algebra operations**: The inclusion of code parallelization as a requirement implies that interprocess communication is needed during linear algebra operations. For instance, parallel matrix-vector multiplication requires interprocess communication in order to complement the job done by each process.

ii) **Communication along process hierarchy**: The establishment of the hierarchy of procedures will require some processes to assume some kind of leadership depending on the layer, where they are located. This will require interprocess communication throughout the hierarchy.

iii) **Communication for coupled information - I**: MPhyScas-S has dealt with coupling dependences between different phenomena already, but in parallel processing this type of dependence can become very complex. For instance, this is the case whenever the interface between two coupled phenomena coincides with a boundary between two components of the mesh partition. That means that vector fields and respective meshes data have to be transferred from one process to the other.

iv) **Communication for coupled information - II**: If two coupled phenomena have different geometric meshes, all components in one mesh partition may be different from all components in the other partition. Thus, there will be a need for interprocess data transfer from one phenomena to the other, whenever coupled quantities have to be computed.

Well-thought distribution schemes and data representation abstractions can eliminate both types of communication for coupled information. This can be done by copying the coupled vector fields and meshes data, to the processes where they are needed. Those copies will be updated whenever needed (communication of type (i) only). Processes will have to be given a larger memory space, but that can be made a minimum. The important thing is that the whole data set of a coupled mesh will not be transferred between two processes when a coupled quantity is to be computed. Thus, only types (i) and (ii) will be needed. They will be called communications of Type-I and Type-II, respectively.

In order to simplify the presentation of the MPhyScas-P's architecture some requirements have to be made: (i) If two or more phenomena are coupled in one geometric entity, then they share the same geometric mesh on that geometric entity. We will not consider transfer of data between different meshes in this work; (ii) All phenomena have to be represented in each process with a nonempty geometric mesh; (iii) Only three hierarchical levels will be considered in this work: Cluster, Machine and Process Levels. We do not differentiate the running processes by their processors in the same machine. Furthermore, we will not divide a process code into Core and Software-Hardware Sub-levels.

Requirements (i) and (iii) are made for simplification of explanation, since if they were not made, some details about mesh partition and distribution and the use of software-hardware procedures would be needed, blurring the center piece of this work. Requirement (ii) is needed because of the lack of a better alternative: we are making an option for a SPMD scheme. A MPMD scheme would require an automatic analysis of the solution algorithm in order to decide what procedure branches could be executed in parallel. Such a solution algorithm analysis is still the subject of an ongoing work.

B. Logical and Topological Views

There are two views of the MPhyScas-P's architecture:

i) **Logical View**: the logic of MPhyScas-P's workflow is the same as the MPhyScas-S', that is, it has the same levels of procedures (Kernel, Blocks, Groups and Phenomena), all relationships between them are preserved and all procedures within each layer are technically the same (besides the fact that data are now distributed). Therefore the relationships among entities in the different levels of MPhyScas-P is also a DAG (direct acyclic graph). Thus, we are able of cloning a suitable modification of MPhyScas-S to all processes in a SPMD scheme. In this sense, one can imagine that MPhyScas-P is MPhyScas-S with distributed data and a hierarchical synchronization scheme (see Topological View below).

ii) **Topological View**: the topology of the procedures in the workflow of MPhyScas-S is implemented in MPhyScas-P in a hierarchical form with the aid of a set of processes, which are responsible for the procedures synchronization. There are three types of leader processes (see Figure 7):

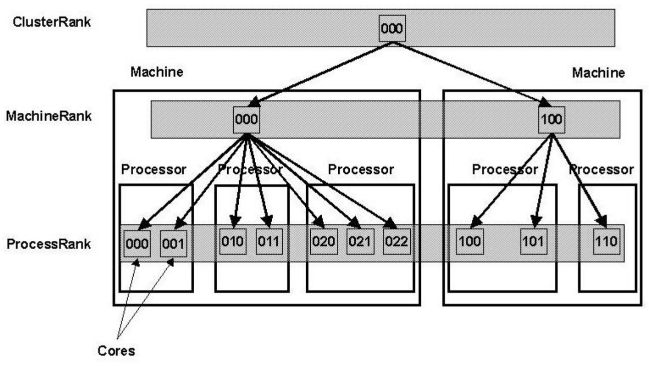

Fig. 7. Hierarchy of the simulator in MPhyScas-P

ii.i) **ClusterRank Process**: it is responsible for the execution of the **Kernel** and to synchronize the beginning and the end of each one of its level's tasks, which requires demands to lower level processes. In a simulation there is only one ClusterRank process (for instance, the process with rank equal to zero in an MPI based system). Figure 8 depicts the relationship between a ClusterRank process and the simulator layers.

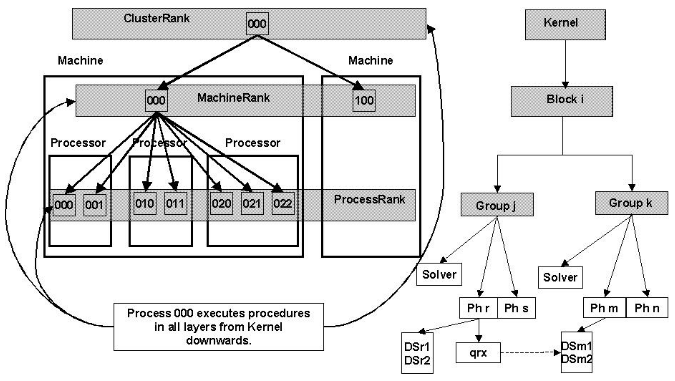

Fig. 8. Layers with procedures executed by clusterRank in MPhyScas-P

ii.ii) **MachineRank Processes**: one process is chosen among all processes running in an individual machine to be its leader. Thus, there is only one MachineRank process per machine. It is responsible for the execution of procedures in the **Block** level and to synchronize the beginning and the end of each one of its level's tasks, which requires demands to lower level processes. ClusterRank is also the MachineRank in its own machine. Figure 9 depicts the relationship between a MachineRank process and the simulator layers.

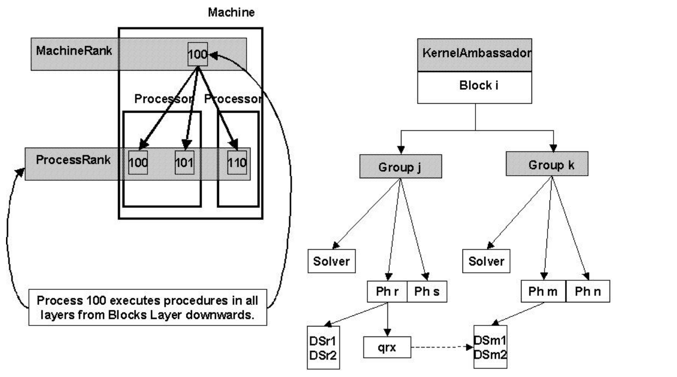

Fig. 9. Layers with procedures executed by machnineRank processes in MPhyScas-P

ii.iii) **ProcessRank Processes**: it is responsible for the execution of the procedures in the **Group** level. The ClusterRank and all MachineRank processes are also ProcessRank processes. Figure 10 depicts the relationship between a ProcessRank process and the simulator layers.

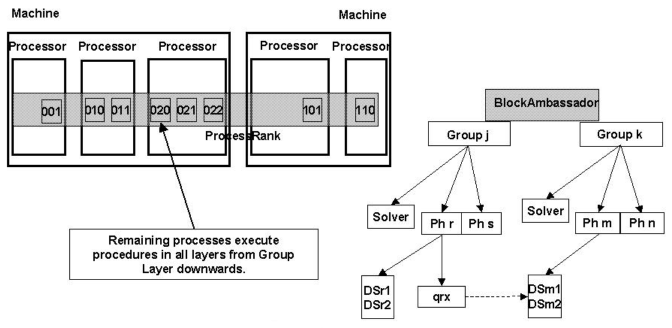

Fig. 10. Layers with procedures executed by processRank processes in MPhyScas-P

Knowing that MPhyScas-S transfer commands from the **Kernel** level down to the **Phenomenon** level in the form of a tree structure, it can be seen that ClusterRank only demands services from all MachineRanks, which only demands services from all of its ProcessRanks. Since the activities in one level returns to the level immediately above after they are accomplished (with the exception of the **Kernel** level) there are natural ways of synchronizing each activity (for instance, using barriers after each demand to the respective lower level has been executed). The heavy computational load is located in the ProcessRank processes. Since all processes are also ProcessRanks and the extra management duties of the leader processes are extremely light, there is no waste of processing power. Furthermore, there is certainly an advantage with the tremendous simplification in the synchronization tasks. Note that the activities in **Group** and **Phenomenon** levels are left for a finer granularity of management. In both levels there are well localized CPU intensive operations, which could be accelerated with a suitable software optimization and the use of hardware devices (for instance, fpga's).

As it has been seen, MPhyScas-P can be considered as MPhyScas-S running in different processes with distributed data. Besides the natural differences between sequential and parallel programs, there is also a specialization of some of the processes, which is important in the synchronization activities. However, when coming to the more demanding parts of the computation, all processes will participate as well. Those parts are coded almost exactly in the same as they are in MPhyScas-S. In what follows we explain the main procedures executed by the preprocessor and by the simulator.

V. Computational Work Load and Flow in MPhyScas-P

In this section we summarize several aspects of the main activities related to the simulator building, the preprocessing of user data and the simulation. In what follows we will describe the following activities: (i) definition and instantiation of the simulator; (ii) Input of simulation data; (iii) Preprocessing; (iv) Simulation execution; (v) Mesh partition and (vi) Visualization.

A. Simulator definition and instantiation

Simulator objects are complex computational entities and are built following a set of user data (actually, meta-data). Simulators in MPhyScas-P architecture do not behave the same in all processes. Therefore, the preprocessing builds simulators able of instantiating different behaviors . Behavior instantiation will be performed depending on the role of each running process, that is, ClusterRank, MachineRank and ProcessRank type processes will behave differently, since they have different management duties. However, in the present implementation, they will perform virtually the same procedures, when it comes to activities at the Group and Phenomenon levels (the most computationally intensive procedures). The definition of a simulator behavior in each process comprises the following activities:

a. ClusterRank (Rank Zero): (i) Interacts with user in order to build/configure the simulator; (ii) Identifies all other processes as either MachineRank or ProcessRank and provides a tag to each one of them; (iii) Format simulator specification data for distribution to each MachineRank process; (iv) Distributes simulator specification data to all MachineRank processes.
b. MachineRanks: (i) Receive simulator specification from ClusterRank; (ii) Format simulator specification data for distribution to its ProcessRanks processes; (iii) Distribute simulator specification data to all its ProcessRanks processes.

c. ProcessRanks: Receive simulator specification from its MachineRank process.
d. All processes: Instantiate simulator (processes from one hierarchical level to another have different simulation instantiation mechanism).

B. Simulation data input

Input of simulation data in MPhyScas-P is exactly the same as for MPhyScas-S (for more information see [4]). However, since processes are specialized - depending on where they are placed in the hierarchy of the simulator - the transfer of simulation data start with the ClusterRank and goes down the hierarchy down to the ProcessRanks. The input procedures are:

a. ClusterRank: (i) Interacts with user in order to input simulation data: (i.1) Geometry; (i.2) Phenomenon types; (i.3) Relation phenomenon $\times$ geometry; (i.4) Quantity to be activated for each phenomenon object; (i.5) Group data;(i.6) Phenomenon data; (i.7) Complementary data for the definition of the preprocessor behavior; (ii) Formats simulation data for distribution to all MachineRanks; (iii) Distributes simulation data to all MachineRanks.
b. MachineRanks: (i) Receive simulation data from ClusterRank; (ii) Format simulation data for distribution to its ProcessRanks; (iii) Distribute simulation data to its ProcessRanks.
c. ProcessRanks: Receive simulation data from its MachineRank.
d. All processes: Instantiate preprocessor object.

C. Preprocessing

Preprocessing is an activity responsible for the building of data structures for the simulation data in a way that can be understood by the simulator. Not only that, of course, because part of the user data is transformed severely, before becoming available for the simulator. Those tasks can be very computationally demanding and can be performed either sequentially - with the result being distributed afterwards - or in parallel. One of such an example is mesh generation. In MPhyScas-P the preprocessing is also especialized, depending on the process type along the hierarchy. In any case, the idea is that the processes in each level will perform part of the preprocessing and will send subsets of raw data together with subsets of already preprocessed data to processes in the lower level. This helps not only load balancing, but also the simplification of procedures.

1) Preprocessing Dynamics: The dynamics of the preprocessing activities can be described through the actions taken at each level of computation:

a. ClusterRank: (i) **If** preprocess is sequential: (i.1) Preprocess whole simulation data including mesh generation and partition; (i.2) Format preprocessed simulation data to all MachineRanks; (i.3) Distribute preprocessed simulation data to all MachineRanks, or **else** (i.1) Preprocess the whole simulation data in parallel with all other processes (communication with other processes depends on the methods used, i.e., mesh generation)
b. MachineRanks: (i) **If** preprocess is sequential: (i.1) Receive preprocessed simulation data from ClusterRank; (i.2) Preprocess a small part of its simulation data; (i.3) Format preprocessed simulation data for distribution to its ProcessRanks; (i.4) Distribute preprocessed simulation data to all its ProcessRanks, or **else** (i.1) Preprocess the whole simulation data in parallel
c. ProcessRanks: (i) **If** preprocess is sequential: (i.1) Receive preprocessed simulation data from rank machine; (i.2) Preprocess a small part of its simulation data; or **else** (i.1) Preprocess the whole simulation data in parallel
d. Notes: (i) The Preprocessor object is actually a very complex machine. It encapsulates a great variety of other objects, which were instantiated following data (choices) given by the user; (ii) This object is specialized depending whether the node is a ClusterRank or a MachineRank or a ProcessRank; (iii) It is not the intention of this paper to go into details about the preprocessing stage. Nevertheless, short explanations about mesh generation and distribution will be needed; (iv) A third party mesh generation in parallel for MPhyScas should require that: (iv.1) ClusterRank starts the process and distributes data to be performed in parallel with all MachineRanks; (iv.2) Then all MachineRanks will process the data a little bit more and then redistribute them to all ProcessRanks; (iv.3) Since ClusterRank and all MachineRanks are also ProcessRanks, the heaviest work will be done after the data is spread among all processes; (v) When the mesh generation is sequential, only ClusterRank executes the mesh generator and then makes the partition and distribution of the mesh; (vi) Being able of using a third party mesh generator is also a requirement for MPhyScas-P. Thus, it is wrapped inside an object, which is also responsible to transfer data in and out of the mesh generator.

2) Preprocessing activities: The following activities comprise the main activities in the preprocessing. For clarity purposes, we assume that the mesh generation is done sequentially by the ClusterRank:

a. ClusterRank: (i) Instantiate Phenomenon objects; (ii) For each Phenomenon object: (ii.1) Build GeomGraph; (ii.2) Build PhenGraph; (ii.3) Activate quantities; (ii.4) Build relationship Phenomenon $\times$ Phenomenon based on coupled quantities; (ii.5) Establish mesh sharing relationship; (ii.6) Instantiate methods; (iii) Relate Phenomenon objects with simulator Groups; (iv) For each Group: (iv.1) Build GroupTask objects and load their data; (iv.2) Build QuantityGroup objects with their GroupTask objects; (iv.3) Instantiate methods; (v) Generate geometric meshes; (vi) Generate phenomenon meshes for each Phenomenon; (vii) Partition each one of the geometric meshes and respective phenomenon meshes among MachineRank processes; (viii) Partition GeomGraphs following geometric mesh partition; (ix) Partition PhenGraphs following the partition of the respective GeomGraphs; (x) Build Phenomenon objects for each partition; (xi) Format data (Group data and Phenomenon data for each partition) to be sent to the MachineRanks processes; (xii) Distribute preprocessed data to MachineRanks processes.

b. MachineRank: (i) receive data from ClusterRank; (ii) Instantiate Phenomenon objects; (iii) For each Phenomenon object: (iii.1) Build GeomGraph; (iii.2) Build PhenGraph; (iii.3) Activate quantities; (iii.4) Build relationship Phenomenon × Phenomenon based on coupled quantities; (iii.5) Establish mesh sharing relationship; (iii.6) Instantiate methods; (iv) Relate Phenomenon objects with simulator Groups; (v) For each Group: (v.1) Build GroupTask objects and load their data; (v.2) Build QuantityGroup objects with their GroupTask objects; (v.3) Instantiate methods; (vi) Recover geometric meshes; (vii) Recover phenomenon meshes for each Phenomenon; (viii) Partition each one of the geometric meshes and respective phenomenon meshes among its ProcessRank processes; (ix) Partition GeomGraphs following geometric mesh partition; (x) Partition PhenGraphs following the partition of the respective GeomGraphs; (xi) Build Phenomenon objects for each partition; (xii) Format data (Group data and Phenomenon data for each partition) to be sent to its ProcessRank processes; (xiii) Distribute preprocessed data to its ProcessRank processes

c. ProcessRank: (i) receive data from its MachineRank; (ii) Instantiate Phenomenon objects; (iii) For each Phenomenon object: (iii.1) Build GeomGraph; (iii.2) Build PhenGraph; (iii.3) Activate quantities;(iii.4) Build relationship Phenomenon × Phenomenon based on coupled quantities; (iii.5) Establish mesh sharing relationship; (iii.6) Instantiate methods; (iv) Relate Phenomenon objects with simulator Groups; (v) For each Group (v.1) Build GroupTask objects and load their data; (v.2) Build QuantityGroup objects with their GroupTask objects; (v.3) Instantiate methods; (vi) Recover geometric meshes; (vii) Recover phenomenon meshes for each Phenomenon; (viii) Build Phenomenon objects

D. Simulation execution

The execution of the simulation requires synchronization in all levels of the hierarchy. We will not describe this mechanism in detail. However, it is important to mention that the execution of tasks at a given level, which requires tasks to be executed by other processes in the lower level, is used as a synchronization point for all processes involved. The main procedures can be viewed below:

a. ClusterRank: (i) Interacts with the user in order to start simulation; (ii) Starts simulation by executing the Kernel driver (it is an object): (ii.1) Whenever the Kernel driver calls the execution of a procedure at the Block level, it should broadcast a message with the needed data to all MachineRanks; (ii.2) Execute its own Block level as requested by the Kernel driver (ClusterRank acts as a MachineRank); (ii.3) Upon the end of the execution of the procedure in the Block level, ClusterRank broadcasts a message to all MachineRanks for synchronization purposes.

b. MachineRanks: (i) Receive message from ClusterRank to execute a procedure in the Block level; (ii) Execute the required procedure; (iii) Whenever a procedure in a Block object demands the execution of a procedure - operation of type BLAS I, II or III or the execution of a GroupQuantity object - at the Group level, it should broadcast a message with the needed data to all its ProcessRanks; (iv) Upon the end of the execution of the procedure in the Group level, MachineRank broadcasts a message to all ProcessRanks for synchronization purposes; (v) At the end of the procedure, MachineRank sends a message answering the synchronization broadcast sent by ClusterRank

c. ProcessRanks: (i) Receive message from its MachineRank to execute a procedure in the Group level; (ii) Execute the required procedure; (iii) At the end of the procedure, ProcessRank sends a message answering the synchronization broadcast sent by its MachineRank.

d. Notes: It is noticeable that the described hierarchical execution in parallel allows also for parallelization schemes of type MPMD (multiple processes multiple data), because it links components in a DAG (direct acyclic graph) structure. The DAG structure allows for automatic and dynamic analysis, load balancing, algorithm partitioning and scheduling of the execution of all its parts on a given set of processors. This is currently being pursued and will be published elsewhere.

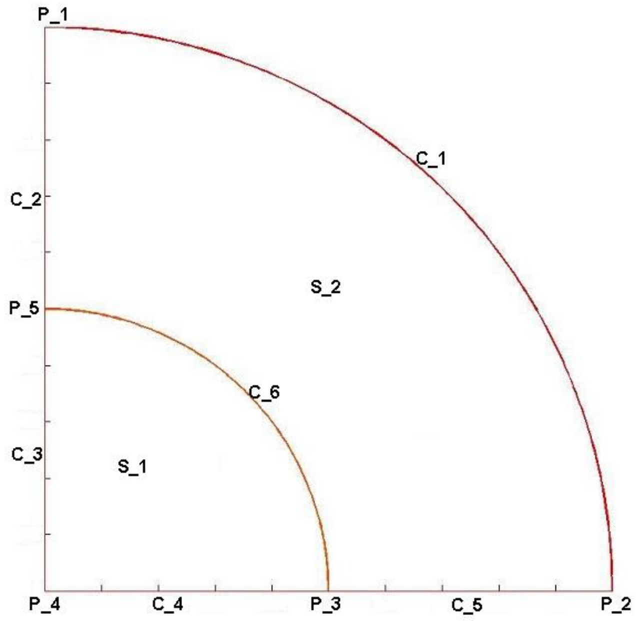

Fig. 11. Geometric domain

E. Further notes

In this subsection we present some notes to further clarify some issues.

1) Mesh partition and distribution: For what follows, consider the geometric domain in Figure 11 and its geometric

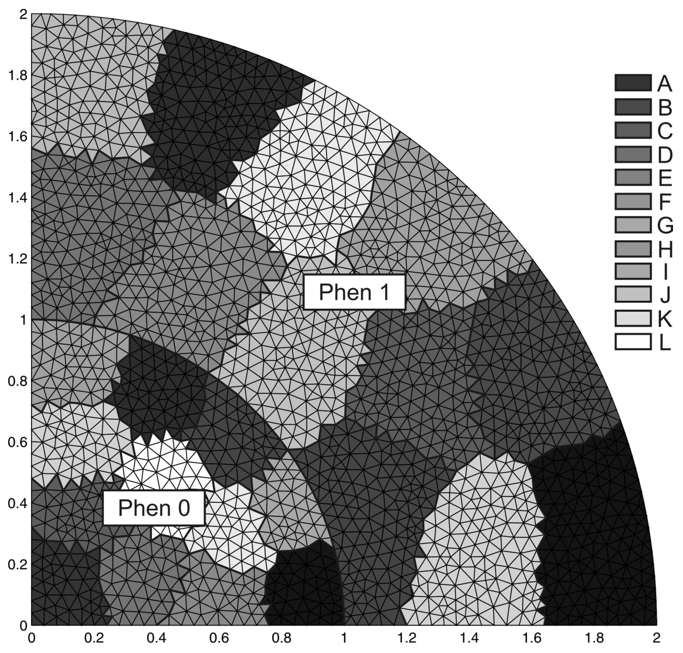

Fig. 12. Partitioned geometric mesh

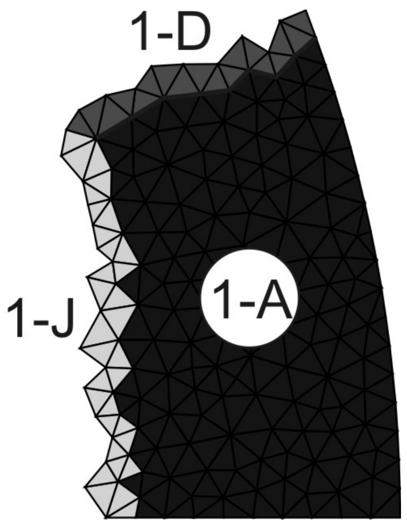

Fig. 13. Partition component in contact with the outer boundary

mesh 12, which is partitioned for twelve processes. Assume that phenomena Ph_1 and Ph_2 act on S_1 and S_2, respectively. Note that both domains were partitioned into twelve parts (processes i, i = A, . . ., L) in order for each process to contain both Phenomenon objects with nonempty meshes.

a. MPhyScas architecture associate procedures (quantity computations and other tasks) to geometric entities [15], [4]. They are performed at the Phenomenon level, but their execution and required parameters are established at the Group level. Those procedures and the geometry are then organized in the form of two graphs, the GeomGraph and the PhenGraph, which have the same structure. However, while the GeomGraph encapsulates a geometric entity (points, curves, surfaces, volumes) at each one of its nodes (GeomGraphNodes), the PhenGraph encapsulates the procedures (called WeakForm) at each one of its nodes (PhenGraphNodes), which are to be computed on the geometric entities of the respective GeomGraphNodes.
b. The partition of the mesh represents a partition of the geometry and thus requires an associated graph partition. The current geometric entities will then be partitioned - after their mesh partition - and new geometric entities will be formed (including those on the mesh interface between two mesh parts), see Figures 13 and 14. Note

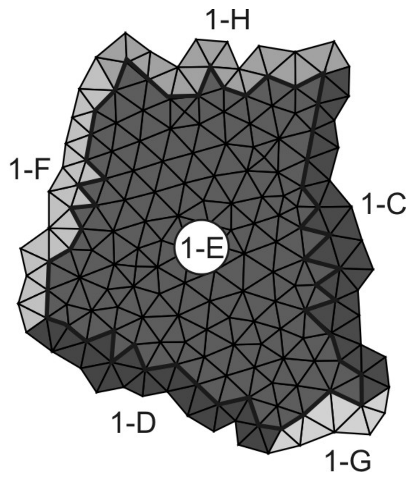

Fig. 14. Partition component without contact with the outer boundary

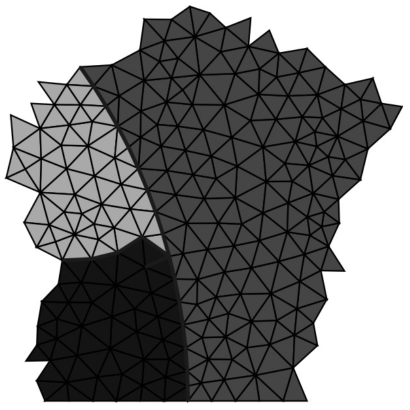

Fig. 15. Neighboring partition components, such that the interface is an existent geometric entity

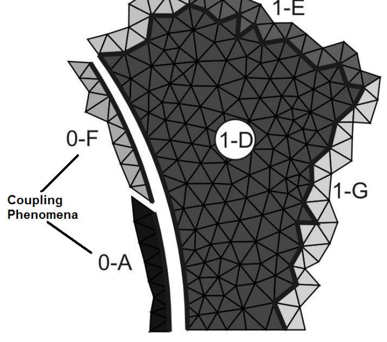

Fig. 16. Added CouplingPhenomenon to the process on one side

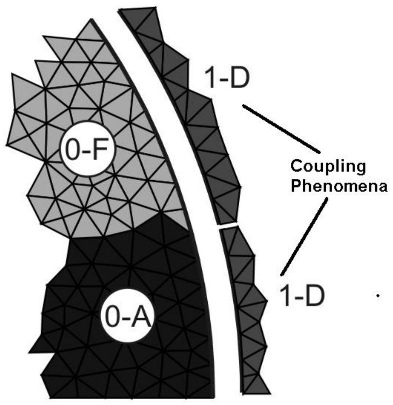

Fig. 17. Added CouplingPhenomena to the process on the other side

that ghost elements are always included in the parts of a partition.

c. The partition of already existent geometric entities will produce new geometric entities (Figure 13), which will inherit all WeakForms from the former. However, the new geometric entities obtained at the interface between two partitions will be given new WeakForms, depending on the solution algorithm implemented in the simulator. Nevertheless, the vector fields restricted to those brand new geometric entities will represent the data to be exchanged between the neighboring processes.

d. Following item (c) above, new GeomGraphs and PhenGraphs will be generated for each partition. It is important that a process P can retrieve the id's of its neighboring processes and the connectivity of the mesh nodes at each interface between P's mesh and its neighboring mesh parts. Those pieces of information can also be localized at the PhenGraphNode associated to the GeomGraphNode, which contains the geometric interface between two mesh parts. Two processes are neighbors if their geometric meshes have a nontrivial intersection.

e. It may happen that an interface between two mesh parts is also a part of an existent geometric entity at the contour of the geometric domain, Figure 15. Suppose that this geometric entity divides the geometry into two regions, where one phenomenon acts on one side and a different one acts on the other (for instance, part A and part D in Figure 15 and 16). In this case, if there is exchange of data between both phenomena during simulation, then, special procedures should take place. This is so because one cannot afford the transfer of coupling data (vector fields and Phenomenon meshes) across processes each time one Phenomenon object needs data from other Phenomenon on the boundary of its geometric domain.

f. In order to tackle the problem described in item (e) above, consider that both meshes were partitioned among all processors in such a way that each process has all Phenomenon objects with nonempty mesh parts. Suppose now that Ph_0 computes a quantity q_a on the interface between S_1 and S_2 (that is, curve C_6), where it needs data from phenomenon Ph_1. Consider process A, which contains two mesh partitions, $A-0$, for Ph_0 and $1-A$ for Ph_1. Consider now process D, which contains two mesh partitions, $0-D$, for Ph_0 and $1-D$ for Ph_1. Note that the interface between $0-A$ and $1-D$ coincides with a part of the curve C_6. Thus, the computation of q_a by process A will need information from Ph_1, which is not in process A (note that $1-A$ is far from that curve). The solution is, then, to add to process A another Phenomenon object called CouplingPhen (in the same Group as Ph_0), containing the copy of the geometric entity, where the coupling occur (interface $0-A$-1$-D$), together with its mesh and related vector fields, Figure 17. Also, this object should know process D's id and handler to its locale in Ph_1 in D. In this way, whenever required, the CouplingPhen object will update - through communication between both processes - only the vector field data from process D related to the geometric part $1-D$. Mesh data (geometric and phenomenon meshes) is already local to process A and need not be transferred.

g. The structure of CouplingPhen is exactly the same as a regular Phenomenon object. The difference is that the instantiation of CouplingPhen is dynamic and does not need data from the user. Also, the coupling information needed in the computation of a coupled quantity (like q_a in the above example) is automatically built from the original information (provided by the user), which linked the computation of q_a by Ph_0 with data from Ph_1 on the curve C_6.

h. Note that geometric interfaces between two parts can be either a point, or a mesh curve, or a mesh surface. This is so because those entities can be shared by more than one partition. Actually, points (in 2-D and 3-D) and mesh curves (in 3-D) can be interfaces between many partitions at the same time. This makes the above story a little more demanding, but we will not go into further details. The main picture is already set.

i. After the mesh partition and distribution is finished, the preprocessing procedures will generate the new GeomGraphs and PhenGraphs for each process. Then, all CouplingPhen objects are instantiated. That is the last thing the preprocessor will do before the simulation. The simulator automatically schedules the updating requests to CouplingPhen objects.

j. Mesh partition and distribution are far more complex when two or more Phenomena objects - defined on the same geometric entity - do not share the same geometric mesh. It is when data transfer between meshes should take place. This case is quite important, but will not be considered here.

2) Visualization and other types of external interfaces: MPhyScas does not provide costume-made visualization machinery for simulation data and results. However, it does provide interfaces to third party visualization software. An interface should be implemented for each new visualization software to be used. Also, the post-processing procedures are implemented as Phenomenon objects, that is, all calculations and format exchange of quantities to be visualized/analyzed are implemented as coupled WeakForm's in Phenomenon objects. The execution of the visualization software can be done through MPhyScas interface, although it can also be done separately. Not only visualization events can be considered. Phenomenon objects can also encapsulate (through their WeakForm's) a variety of different types of interferences in the simulation. For instance, interruptions and coupling between simulation and laboratory experiments can be implemented with this strategy. The simulation algorithm will dictate when and where in the simulation those interferences will become active.

VI. Conclusions

We presented the architecture of MPhyScas-P, a framework aimed at supporting the automatic development of high performance simulators for multi-physics problems. This architecture inherits from MPhyScas-S (the sequential version) all its workflow representation, with the obvious difference that MPhyScas-P is distributed in a hierarchical way. Although MPhyScas-S has already a fully functional prototype, MPhyScas-P has a prototype (using MPI) currently being tested. Besides those qualities that MPhyScas-S has already demonstrated (strong reusability, maintainability, adaptability and correctness), MPhyScas-P provides also another nice feature: due to the DAG (direct acyclic graph) structure of its workflow, its code can be dynamically analyzed and reconfigured in such a way that MPMD schemes could be used. At last but not least, it is important to notice that we are dealing with very complex problems, with very complex solution algorithms. We think that if MPhyScas-P would be able to alleviate the burden of programming and changing code for those types of problem, our task would be fulfilled. We are currently building a graphic user interface coupled to a DBMS in order to manage the use of components for MPhyScas-S and MPhyScas-P and are planning in using an interface description language in order to describe all interfaces of components. One candidate being considered is SIDL from the Common Component Architecture (CCA) [13].

References

[1] F. C. G. Santos, E. R. R. J. Brito, and J. M. A. Barbosa, "Simulao do problema de evoluo do dano em uma barra elasto-viscoplstica com acoplamento termomecnico empregando grafo de interface genrica (gig)," 7° *Congresso Iberoamericano de Engenharia Mecnica*, 2005.

[2] F. C. G. Santos, E. R. R. J. Brito, and J. M. A. Barbosa, "Coping with data dependence and sharing in the simulatin of coupled phenomena," *International Congress on Computational and Applied Mathematics - ICCAM*, 2006.

[3] F. Santos, M. Lencastre, and M. Vieira, "Workflow for simulators based on finite element method," *Proceedings of the International Conference on Computational Science (ICCS)*, 2003.

[4] F. Santos, E. R. R. J. Brito, J. M. A. Barbosa, J. M. B. Silva, and I. H. F. Santos, "Toward the automatic development of simulators for multi-physics problems," *International Journal of Modelling and Simulation for the Petroleum Industry*, vol. 1, no. 1, 2007.

[5] M. Lencastre, *Conceptualisation of an Environment for the Development of FEM Simulators.* Doutorado em ciências da computação, Universidade Federal de Pernambuco, Recife, Pernambuco, 2004.

[6] F. Santos, M. Lencastre, and I. Rodrigues, "Fem simulator based on skeletons for coupled phenomena," *Proceedings of the 2nd Latin American Conference on Pattern Languages of Programming*, 2002.

[7] F. Santos and M. Lencastre, "An approach for fem simulator development," *Journal of Computational ans Applied Mathematics*, 2006.

[8] R. Niekamp, "Software component architecture," *http://congress.cimne.upc.es/cfsi/frontal/doc/ppt/11.pdf.*

[9] *http://www.comsol.com/.*

[10] S. Parker, "A component-based architecture for parallel multi-physics pde simulation," *Future Generation Computer Systems*, no. 22, p. 204216, 2006.

[11] A. G. L. G. M. J. R. T. S. E. S. J. Goodale, T., "The cactus framework and toolkit: Design and applications," *Vector and Parallel Processing - VECPAR '2002, 5th International Conference, Springer*, 2003.

[12] H. C. Edwards, "Sierra framework version 3: Core services theory and design," *Sandia National Laboratory, report SAND2002-3616*, November 2002.

[13] "Common component architecture home page," *http://www.cca-forum.org/.*

[14] T. S. G. M. Graybill, B., "Hpc application software consortium summit - concept paper," *http://www.cct.lsu.edu/ gallen/Reports/HPCASC_March2007.pdf*, March 25-26, 2008.

[15] F. C. G. Santos, E. R. R. J. Brito, and J. M. A. Barbosa, "Phenomenon computational coupling relationship between phenomena on multi-physics simulation," *20th European Conference on Modelling and and Simulation*, 2006.