Fourth International Conference on Advanced COmputational Methods in ENgineering (ACOMEN 2008)
Editors: M. Hogge, R. Van Keer, L. Noels, L. Stainier, J.-P. Ponthot, J.-F. Remacle, E. Dick
©University of Liège, Belgium, 26-28 May 2008

An Architecture for the Automatic Development of High Performance Multi-Physics Simulators

Félix C. G. Santos[1], José Maria A. Barbosa[1], José Maria Bezerra[1]
and Eduardo R. R. de Brito Junior[1]

[1] Federal University of Pernambuco - Department of Mechanical Engineering
Rua Acadêmico Hélio Ramos, s/n - Recife - PE 50740-530 - Brazil

e-mails: *flxcgs@yahoo.com.br*, {*jmab@ufpe.br, zemaria@ufpe.br, errbj@yahoo.com.br*}

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; phenomena-phenomena and phenomena-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 supports the development of 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 both types of simulators, i.e., sequential and parallel. We take an advantage of the architecture in layers of MPhyScas in order to define a hierarchical parallel computation scheme in such a way that the need for communication between processes is automatically identified and objects are built in order to fulfill that need. Also, the hierarchy provides a natural way of defining data structures and their access dynamics for all memory levels, providing simpler ways of dealing with non-uniform memory access patterns. Some preliminary results obtained with a prototype applied to the simulation of hydrogen fragilization of steel alloys will be shown and analyzed.

1 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 (Kernel in MPhyScas) 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 (where Blocks live in MPhyScas) 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 (where Groups are located in MPhyScas) 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 (where Phenomenon objects are located in MPhyScas) 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 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, two types of communication between processes during a simulation are identified and patterns are defined for their representation. All those aspects will be described with some detail and preliminary results will be shown and analyzed.

This work is organized as follows: first comments on some important related work are provided in order to build a context for this article. Next the architecture of MPhyScas-P is described, and some comments are drawn. The purpose of this article 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.

2 Related Work

Reading the next section before this one may help understanding many issues herein considered. 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 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. Important works of class (i) are the Component Template Library (CTL) [5] and the Comsol [6]. CTL provide abstractions that support the building of solutions algorithms for loose and tight coupling. It is an implementation of the component concept with an RMI semantic similar to CORBA or Java-RMIcomponents, 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 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, neither one of them provides structural guidance for simulators, leaving that task to the user. Of course, there are several other works, which are not mentioned here in order to make this description more objective.

In the class of frameworks (class (ii)), one can find powerful packages, such as Uintah [7], Cactus [8] and Sierra [9]. Since the architecture of MPhyScas is closer to Sierra's architecture, we will comment 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 characterize them as having a deep interoperability system. Uintah uses CCA (Common Component Architecture) for designing and describing components interfaces. It does not have a pre-defined structure for simulators. Thus, in order to provide framework functionallity 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 [7].

The following nice summary of Cactus structure can be found in [10]: "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".

Sierra framework provides a structural guidance in layers composed of (from top to botom) **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 algorithms. A Mechanics implements procedures related to a specific physics - defined on a subset of its Region's mesh - and its MechanicsInstances is responsible for an atomistic operation defined on a subset of its Mechanics' mesh. A Mechanics may use another set of Mechanics, building one more layer. This may be used in multiscale computations, where a lower level Mechanics is used to compute constitutive data for a MechanicsInstance of a its parent [9]. If a Mechanics A needs data from another Mechanics B (provavly in another Region), the core services of Sierra provides means to transfer mesh-dependent data from B's mesh to A's mesh. The result is then stored in the Region of Mechanics A. The SIERRA Framework core services also 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 importante differences:

i) MPhyScas is strongly concern-oriented, instead of procedure-oriented or context-oriented. Concerns are more easily mapped into requirements and architectures. Concerns are related to the fundamentals of 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 build to satisfy a nested set of concerns (functional and non-functional) related to the numerical approximation to solutions of partial differential equations.

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 responsabilities of the layer of Blocks.

iii) MPhyScas's Block is quite similar to Sierra's Procedure in their generality. 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, MPhyScas limited the depth of this layer in a slice, where activities related to a time step for a set of sets of phenomena are defined and articulated. However, in MphyScas each Block also has the responsability 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. The motivation for the Group in MPhyScas is based on a set of Phenomenon objects and their data, which participate in linear monolithic algebraic systems. Thus, all data needed to assemble and solve those algebraic systems are in the Group as well as in the Region. However, a Group does not know anything neither about geometry domains nor about meshes. Those pieces of information are located in the respective phenomenon. There are special data structures that allow two phenomena 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 is described by user data. The location of matrices and vectors in the Group was motivated by the location of linear solvers in the Group. Transfer of vector fields from one mesh to the other is provided by the framework in Sierra and it is performed by a phenomenon in MPhyScas. All dependencies between Phenomenon objects are resolved between them. 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. There are other differences, but the cited references is able of providing further information.

v) Mechanics in Sierra encapsulate procedures related to a particular phenomenon. 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 yet support p and h-p adaptivity. This certainly would complicate transfer procedures and the way coupled vector fiels are used in Sierra.

Both architectures (MPhyScas' and Sierra's) were independently developed . The first definition of the MPhyScas' architecture was published in 2001. Nevertheless, they present similar structures, which gets deeper in MPhyScas-P, where the hierarchical architecture of MPhyScas is strongly used. 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), where almost all its members are involved.

3 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 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; (iii) The hierarchy allows for the encapsulation of services and procedures, making it easier the exchange of 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.

Next we provide a description of how interprocess communication is considered in MPhyScas-P architecture. The architecture of MPhyScas-S [4] has satisfactorily solved the main problems related to data dependence and sharing between phenomena for the sequential processing. It is also been able of representing solution algorithms in such a way that the entire simulator can be reconfigured with a minimum amount of reprogramming (maximum amount of reuse). However, an essential difference, when considering parallel processing, is the need for interprocess communication, 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 several 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 mesh 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 process 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 mesh 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; (ii) All phenomena should 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.

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 1):

(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 2 depicts the relationship between a ClusterRank process and the simulator layers;

(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

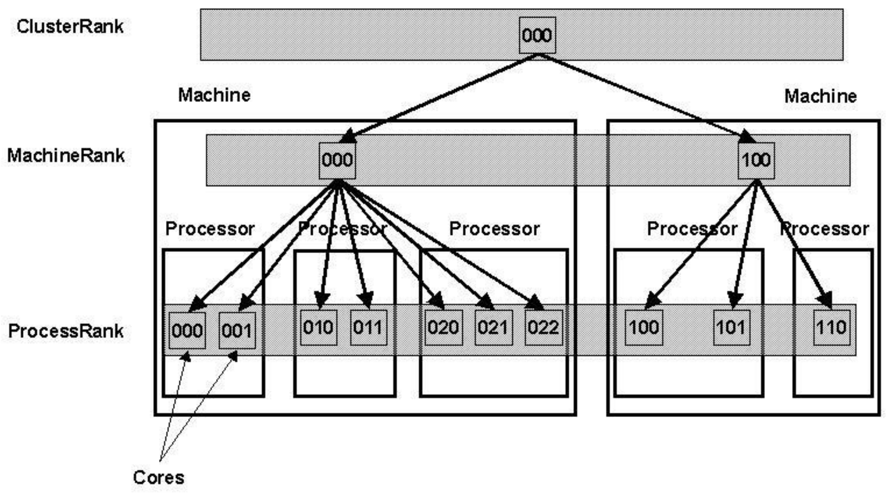

Figure 1: Hierarchy of the simulator in MPhyScas-P

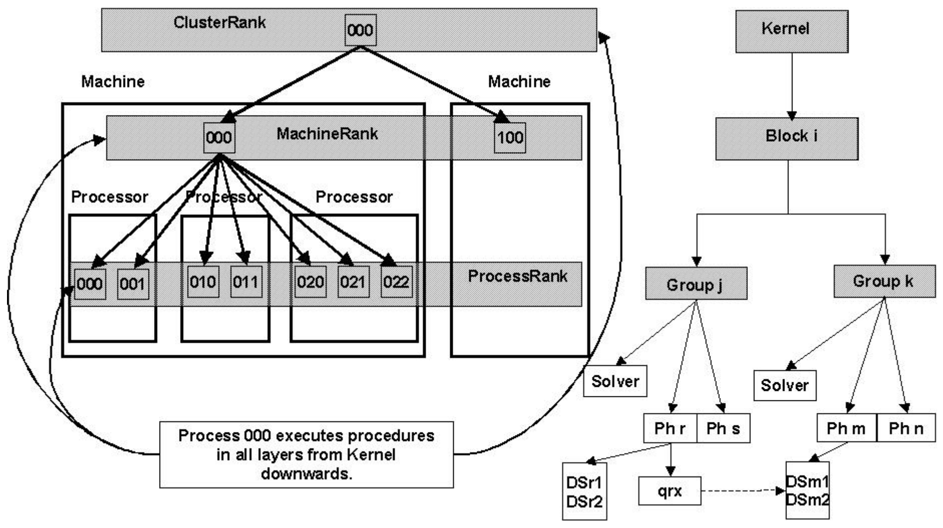

Figure 2: Layers with procedures executed by clusterRank in MPhyScas-P

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 3 depicts the relationship between a MachineRank process and the simulator layers;

(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 4 depicts the relationship between a ProcessRank process and the simulator layers.

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 finished (with the exception of the **Kernel** level) there are natural ways of synchronizing each activity. 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).

In what follows we explain the main procedures executed by the preprocessor and by the simulator.

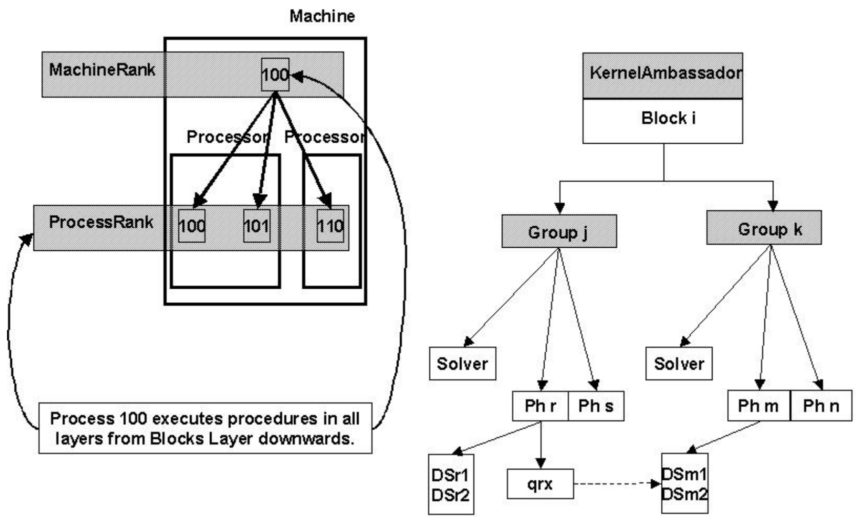

Figure 3: Layers with procedures executed by machnineRank processes in MPhyScas-P

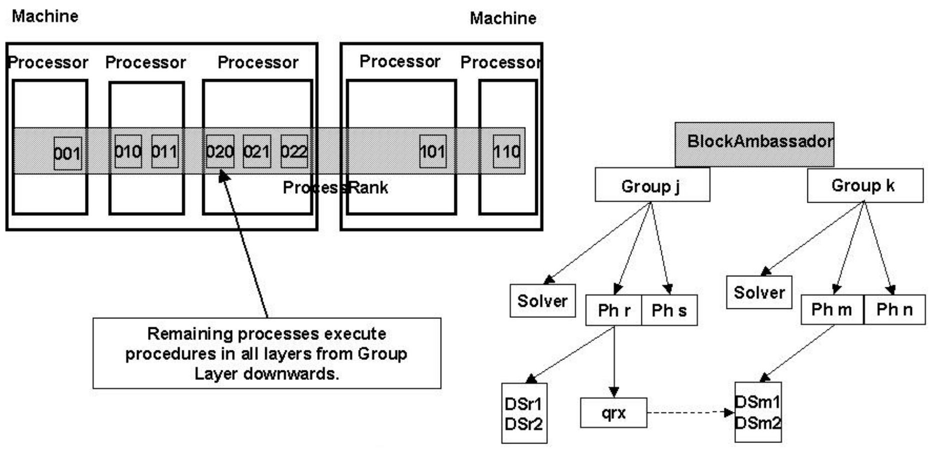

Figure 4: Layers with procedures executed by processRank processes in MPhyScas-P

4 Preprocessing and Simulation

In this section we summarize several aspects of the main procedures related to the simulator building, the preprocessing of user data and the simulation.

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 that are able of instantiating different behaviors. Behavior instantiation depends on the role of each running process, that is, ClusterRank, MachineRank and ProcessRank. However, in the present implementation, they will perform virtually the same procedures, when it comes to activities at the Group and Phenomenon levels. 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 [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) *Preprocess*: Preprocessing is an activity responsible for data structures building to store simulation data in its internal representation. 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 be performed either sequentially (by one process) with the result being distributed afterwards, or in parallel. In MPhyScas-P the preprocessing is also specialized, depending on process type across the hierarchy. In any case, the idea is that the processes in the upper two levels will perform part of the preprocessing and send the relevant results together with a subset of the raw data to be processed in the lower level. This helps not only load balancing, but also the simplification of procedures. The main procedures are:

   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 ranks machine; (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 one. 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.

D) *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 $\times$ 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 $\times$ 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

E) *Simulation execution*: The execution of the simulation requires synchronization in all levels of the hierarchy. 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.

5 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 a 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 will be able to alleviate the burden of programming and changing code for those types of problem, our task will 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 such as SIDL from the Common Component Architecture (CCA).

References

[1] F. C. G. Santos, E. R. R. JR. 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. JR. 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. JR. 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*, 1(1), 2007.

[5] Rainer Niekamp. Software component architecture. *http://congress.cimne.upc.es/cfsi/frontal/doc/ppt/11.pdf.*

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

[7] S.G. Parker. A component-based architecture for parallel multi-physics pde simulation. *Future Generation Computer Systems*, (22):204216, 2006.

[8] Allen G. Lanfermann G. Mass J. Radke T. Seidel E. Shalf J. Goodale, T. The cactus framework and toolkit: Design and applications. *Vector and Parallel Processing - VECPAR '2002, 5th International Conference, Springer*, 2003.

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

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