DESIGN AND IMPLEMENTATION OF MULTIPHYSICS SIMULATORS USING A COMPUTATIONAL FRAMEWORK

Felipe Augusto Cruz

Félix Christian Guimarães Santos

José Maria Andrade Barbosa

felipe84cruz@hotmail.com

flxcgs@yahoo.com.br

andrade.jmab@gmail.com

Federal University of Pernambuco

Rua Acadêmico Hélio Ramos, s/n, 50740-530, Pernambuco, Recife, Brasil

Abstract. *This work intends to show that the multi-physics simulations can be decomposed into four hierarchical layers, facilitating their implementation. These simulations involve the coupling between phenomena using a framework called MPhyScas. To compute approximate solutions involving the problems that deal with coupled phenomena, this work is based on solution methods that decompose the global problem into several sub-problems using the operator decomposition method. These sub-problems, because they are simpler, allow the use of simpler algorithms to obtain their solutions, however the organization of the simulator becomes more complex, being necessary the use of a manipulator to restructure the global problem, then the use of MPhyScas as a guide and structural support contributes greatly to the development of these simulators. The developed methodology is presented through a problem of elastoplasticity under the diffusion of hydrogen, allowing to illustrate the main advantages of the decomposition, which are: ease of organization of simulators; separation of concerns related to the various algorithms used to solve the problem; adequate details of the representation of the various tasks required in the simulation of the problem. This work demonstrates that the hierarchical view of coupled phenomena simulators can be very relevant to its implementation since it facilitates: the use of a computational infrastructure to reduce development time; reuse software; cooperative work; make the final result more understandable and maintainable.*

Keywords: *Simulation, Multi-physics, Finite Element Method, Operator Splitting Method*

1 COUPLED PHENOMENA

The term multi-physics can be defined as a qualifier for a set of interacting phenomena in space and time. These phenomena are usually of different natures 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 dependence may occur in any geometric part, where both phenomena are defined. Another type of data dependency is the case where, after discretization, two or more phenomena are defined in the same geometric component and share the geometric mesh (discretization data). Multi-physics and multi-scale problems are difficult to simulate and the building of simulators for them tend to be very demanding in terms of time spent on their design and implementation (Santos et al., 2007). The main reason is the lack of reusability and maintainability.

Each phenomenon has a law of behavior, which is usually described by partial differential equations with initial and boundary conditions. This boundary value problem depends on a set of vector fields and constitutive parameters that represent the state of the phenomenon and the material where it is applied. Rephrasing a previous definition in a more concise way, if the behavior law of a phenomenon depends on data from other phenomena, then one can say that they are coupled phenomena.

The finite element method (FEM) discretization process defines approximate vector fields in finite dimensional spaces for each phenomenon (Ross, 1996). Thus, each of these discrete vector fields can isomorphically be represented by vectors in R^n and the respective discrete laws of behavior are cast into systems of algebraic equations (either linear or nonlinear) Simo et al., 1992. Since there were several coupled behavior laws, the correspondent systems of algebraic equations are also coupled.

2 OPERATOR SPLITTING METHOD

If the algebraic systems obtained by the discretization process are solved as a single system, it is said that the problem was solved monolithically. If they are solved in partitioned form, it is said that there was a splitting of the operator. The latter class of methods allows for solving coupled problems through a sequence of solutions for simple problems (partial problems), for which solution methods exist and have already been tested (Shaidunov et al., 1983). This class of methods almost always uses iterative processes involving solutions of partial problems. These iterative loops may converge or not, depending on how the partial problems are defined.

In order to understand some of the advantages associated with the operator splitting methods, it suffices to consider the case in which the monolithic system is a linear algebraic system with size $k \cdot n$ where k is the number of phenomena and n is the size of each phenomenon algebraic system (assuming all of them have the same size). The following difficulties which may occur in the solution of the problem in its monolithic form may indicate advantages in employing operator splitting methods:

- the representation of the couplings in the discretized system may be difficult or even impossible due to, for example, higher order derivatives present in the behavior laws;

- Monolithic systems may lead to poorly conditioned matrices, requiring specially designed preconditioners and solvers. Moreover, its solution process may be more costly and complex than a simpler and more accurate algebraic iterations involving solutions to smaller problems;
- in the case of non-linear systems, the monolithic Jacobian matrices can be complex and costly to generate and may present conditioning features not yet known;
- despite simulators are larger programs in the case of operator splitting methods, they are considerably simpler and more understandable with greater possibilities for reuse of existing software components;
- the convergence of discrete solutions is a feature to be checked, since for monolithic solutions the proof of convergence is - in principle - easier to be obtained (not considering the computational cost involved), while for solutions obtained by operator splitting methods that property can not be ensured without complex analysis or very stringent requirements.

For illustration purposes, consider a system that has three phenomena mutually interacting with behavior laws L_a, L_b, L_c, respectively, represented in terms of vector fields of their own and from other phenomena, such as:

$$\begin{cases} L_a\left(u_a, u_b, u_c\right) = 0 \\ L_b\left(u_a, u_b, u_c\right) = 0 \\ L_c\left(u_a, u_b, u_c\right) = 0 \end{cases} \tag{1}$$

Suppose that, through a FEM discretization process, these laws of behavior are transformed into the following systems of algebraic equations.

$$\begin{cases} S_1\left(\mathbf{U}_a, \mathbf{U}_b, \mathbf{U}_c\right) = \mathbf{0} \\ S_2\left(\mathbf{U}_a, \mathbf{U}_b, \mathbf{U}_c\right) = \mathbf{0} \\ S_3\left(\mathbf{U}_a, \mathbf{U}_b, \mathbf{U}_c\right) = \mathbf{0} \end{cases} \tag{2}$$

where $\mathbf{U}_a$, $\mathbf{U}_b$ and $\mathbf{U}_c$ are vectors in $\mathbf{R}^n$. These systems can be considered as a monolithic one:

$$S(\mathbf{U}) = \mathbf{0}, \tag{3}$$

where $\mathbf{U} = \begin{bmatrix} \mathbf{U}_a & \mathbf{U}_b & \mathbf{U}_c \end{bmatrix}$, and can be solved without distinction between the parts that it is consisted of. Solution algorithms for solving this problem may employ, for example, the Newton-Raphson method (in case they are nonlinear), which will lead to the following form:

Algorithm 1 Newton-Raphson Method

Initialize: $\mathbf{U_K}$
while (Convergence criteria are not satisfied) **do**
  Compute jacobian $\mathbf{R}(\mathbf{U_K})$ and residual $\mathbf{S}(\mathbf{U_K})$
  Solve linear system $\mathbf{R}(\mathbf{U_K}) \cdot \Delta\mathbf{U} = -\mathbf{S}(\mathbf{U_K})$
  $\mathbf{U_K} += \Delta\mathbf{U}$
  Compute convergence data
end while
if (Converged) **then**
  $\mathbf{U} = \mathbf{U_K}$
else
  Exit with error statement
end if

Suppose that now we want to perform this calculation using an operator splitting method using three coupled algebraic systems:

$$\begin{aligned} \mathbf{S}_a^*(\mathbf{U}_a, \bar{\mathbf{U}}_b, \bar{\mathbf{U}}_c) &= 0 \\ \mathbf{S}_b^*(\bar{\mathbf{U}}_a, \mathbf{U}_b, \bar{\mathbf{U}}_c) &= 0 \\ \mathbf{S}_c^*(\bar{\mathbf{U}}_a, \bar{\mathbf{U}}_b, \mathbf{U}_c) &= 0 \end{aligned} \tag{4}$$

where the variables ($\bar{\mathbf{U}}$) are assumed to be known for the respective algebraic system.

A solution algorithm for this system would be:

Algorithm 2 Example: Solution Algorithm

Initialize: $\bar{\mathbf{U}}_b$, $\bar{\mathbf{U}}_c$
while (Convergence criteria are not satisfied) **do**
  Solve $\mathbf{S}_\mathbf{a}^* = \mathbf{0}$ using algorithm A, obtaining $\mathbf{U_a}$
  $\bar{\mathbf{U}}_a = \mathbf{U}_a$
  Solver $\mathbf{S}_\mathbf{b}^* = \mathbf{0}$ using algorithm B, obtaining $\mathbf{U_b}$
  $\bar{\mathbf{U}}_b = \mathbf{U}_b$
  Solver $\mathbf{S}_\mathbf{c}^* = \mathbf{0}$ using algorithm C, obtaining $\mathbf{U_c}$
  $\bar{\mathbf{U}}_c = \mathbf{U}_c$
  Evaluate global convergence criteria
end while
if (Converge) **then**
  Accept $\mathbf{U}_a$, $\mathbf{U}_c$, $\mathbf{U}_c$.
else
  Exit with error statement
end if

3 HIERARCHICAL VIEW OF THE SIMULATOR

As in the context of only one physics, simulators for multiphysics problems are applications that may provide many options for simulator configuration to the users: global solution

algorithms; local solution algorithms, linear solvers, discretization methods and so on Santos et al., (2006).

Using the operator splitting method for multiphysics problems, it is known that a specific solution method is applied to a subset of phenomena. The articulation of the execution of these local algorithms within an iteration can be expressed in hierarchical levels in the form of layers, where each layer level is responsible for a particular set of tasks. For example, the higher level layer is responsible for defining the iterative loops in time and iterations involving global solution algorithms. Thus, tasks to be solved at each time instant are triggered by this layer and may require services to be performed by the layer immediately below, which is the layer where local solution algorithms should be defined. Usually, as already illustrated, local solution algorithms make use of many BLAS (Basic Linear Algebra) operations - including linear solvers - with vector fields shared by a subset of phenomena. Since those two previous layers provide more of control structures than arithmetic operations, there is a need for a third layer dedicated to the storage of vector fields and the actual execution of BLAS and linear solvers operations. Finally, a fourth layer may be defined in order to separate the concerns of each phenomenon. Since a subset of phenomena may share algebraic structures (scalars, vectors and matrices), the knowledge about what should be put by each phenomenon on those structures should located appropriately in separate instantiations of specific abstract classes. Thus, we end up with four layers in the simulator hierarchy. Obviously, a larger number of layers may be defined and there might be advantages for that. However, that might as well produce an unnecessary increase in the complexity of setting up a simulator, which is certainly a disadvantage. It is important to point out that the number of tasks to be defined in each layer increases exponentially from top to bottom of the hierarchy.

In the field of multiphysics problems it is important that simulators could be reconfigured and that some of its parts could be reused for the purpose of improving time cost and accuracy of simulations. For instance, reconfigurations may be desirable whenever there is a need for exchanging linear solvers or solution algorithms. Reuse occurs when pieces (software components) of a previously built simulator are used in building a new one. A computing infrastructure should possess efficient abstractions in such a way to minimize the effort and complexity associated with the reconfiguration processes and reuse of software components (Santos et al., 2003). Another aspect of multiphysics simulator development is a need for cooperative work, because different phenomena often require different expertises. Therefore, software components can be developed by different teams in parallel - following previously agreed interfaces - in such a way that their integration into a simulator may be achieved with high productivity levels and excellent quality of the final product.

4 ABSTRACTIONS OF MPHYSCAS

Based on the above, an architecture was designed for a computational infrastructure devoted to support the development of multiphysics simulators that should have the following characteristics: flexibility (changing made easier), extensibility (adding new functionalities), reusability (saving effort), among others. MPhyScas is the name of an implementation of this infrastructure Lencastre (2004) that presents a set of abstractions and a pattern language to support the development of multiphysics simulators decomposed into four hierarchical layers of activity (see Figure 1).

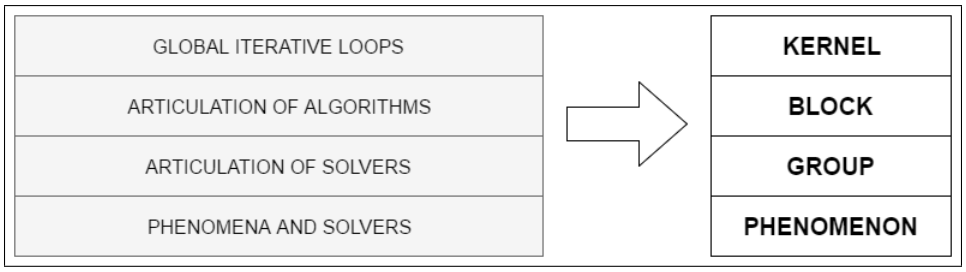


Figure 1: Computational representation of simulator layers

Then, computational representations have been implemented for the layers Kernel, Block, Group and Phenomenon. The programming language used was C++. Besides the fact that it is widely used for scientific software production it supports object-oriented design, programming and has an extremely active community of users. The definition of those layers may be cast as below.

Kernel: It is the top layer, is consisted of a single object (Kernel) that controls all the simulation and is responsible for the articulation of solution algorithms in global loops and iterations. The kernel has a unique algorithm, with a single command for its execution.

Block: It is the second layer in the simulator hierarchy, consists of one or more independent objects (Block) and receives commands directly from Kernel to execute standardized activities related to the execution of its algorithms.

Group: It is the third hierarchical layer and consists of one or more objects independent from each other and which belong to one and only one object Block. Each Group object is able of storing their own data structures with scalar, vectors and matrices and performing operations with them. These objects are also responsible for solving linear algebraic systems.

Phenomenon: this layer consists of independent objects, which belong to one and only one Group object. These Phenomenon objects produce and perform operations with quantities of their own responsibility. These tasks may depend on meshes as well as quantities belonging to other Phenomenon objects (in the case of coupled quantities).

With the setting up of each layer, the simulator can be represented in a tree format, as shown in Figure 2, where we can observe the interaction between hierarchical levels. The capital letters K, B, G and Ph are related to each one of the layers from top downwards.

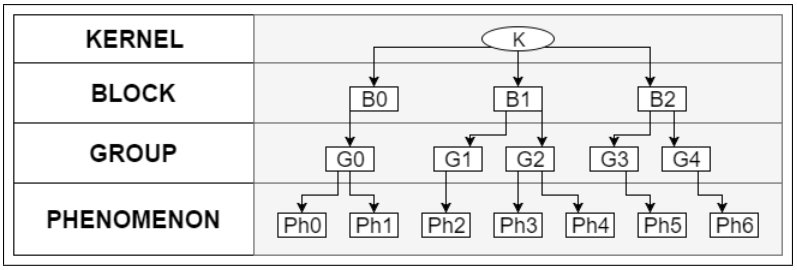


Figure 2: Hierarchy of layers

Remember that $B0$, $B1$, $B2$ are distinct Block objects that belong to the Block Level. Each one is responsible for a different set of procedures (algorithms). Taking aside the specificities of each layer, all objects Kernel, Block, Group and Phenomenon are nodes in the simulator tree.

When an MPhyScas simulator is executed together with specific simulation data, the first task it performs is to ask the Kernel to execute its algorithm (defined by *Alg* in Figure 3), which will in turn ask its Blocks to perform tasks in a prescribed order among other activities it was programmed to perform. The only task the Kernel can ask a Block to do is to execute one of its algorithms (see Figure 3). An example can be seen below:

Algorithm 3 Example: Kernel Algorithm

InitializeTime
Block $B0$: execute $B0(A2)$ algorithm
Block $B1$: execute $B1(A0)$ algorithm
while (time $<$ Interval) **do**
  Block $B0$: execute $B0(A6)$ algorithm
  Block $B0$: execute $B0(A0)$ algorithm
  Step in time
  Block $B1$: execute $B1(A3)$ algorithm
end while

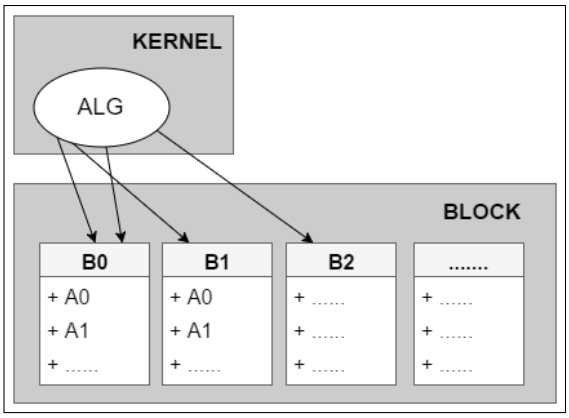


Figure 3: Kernel Algorithm Diagram

Each Block object may have one or more algorithms. The Kernel can request the execution of any algorithm to a particular Block, which is the only form of direct interaction between them. An algorithm for a particular Block may require the execution of a GroupTask to any of its Group objects. Below is an example of how this can be done:

Algorithm 4 Example: Block Algorithm - Block $B0$: algorithm $A0$

Group $G0$: execute your $G0$ GroupTask
Group $G0$: execute your $G3$ GroupTask

Note that, in Figure 4, the algorithm (*A0*) of the Block (*B0*) requests the execution of some task to its Groups (*G0*) and (*G1*), which is made asking for the execution of an object of the type GroupTask. A GroupTask object stores a list of references to Phenomenon objects and respective set of task codes (called execCodes).

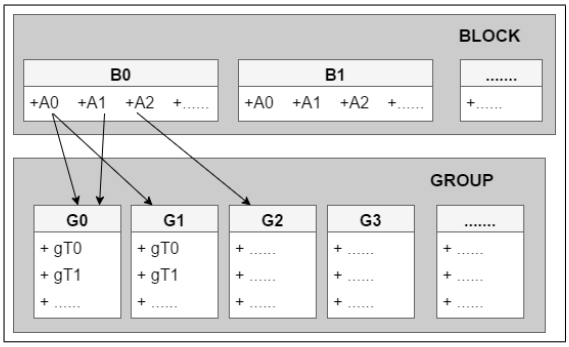


Figure 4: Block Algorithm Diagram

When is required to do so, a Group object will execute a GroupTask object, which in turn will request each one of the Phenomenon objects in its list to perform the respective set of

execCodes. Among these tasks there may be some activities directed to the Group itself (such as Blas operations or solutions of linear algebraic systems), which are carried out by a special type of Phenomenon object, which is gPhen. Each Group object has an unique gPhen object, which becomes its representative in the Phenomenon layer. An example of this operation is shown in Figure 5.

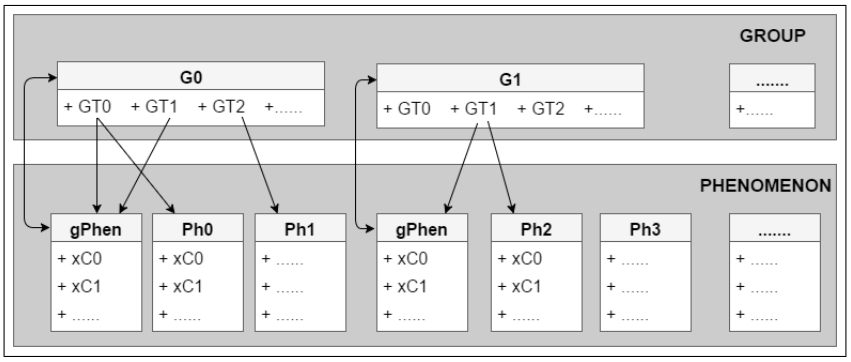


Figure 5: GroupTask Diagram

Algorithm 5 Example: Group Algorithm - Group $G0$: GroupTask $G0$

Request to the $Ph0$ object to execute the $xC3$ execCode
Request to the $Ph1$ object to execute the $xC0$ execCode
Request to the gPhen object to execute the $xC2$ execCode

Figure 5 shows the interaction between a Group and Phenomenon objects.

It remains to the Phenomenon objects the execution of their execCodes, as soon as requested by their Group object. An execCode is actually, an ordered set of codes for well defined tasks called WeakForms. A WeakForm object can encapsulate specific activities for a Phenomenon object, such as computation of matrices and vectors at the level of finite elements and their assemblies into global structures. Those global structures are stored on the Group level. Other examples of activities are: introduction of boundary conditions; data recording for viewing; specific quantities and calculations that depend on the mesh and vector fields concerning the Phenomenon object in question, among others.

Thus, with the structural guide provided by MPhyScas a simulator can be built by defining the objects involved in the interactions between the four layers, as described above, independently of specific simulation data. Simulation data are uncoupled from the simulator definition and building and can then be introduced at a later time. Therefore there are two sets of information that should be supplied separately: Data for the construction of the simulator, and data for defining a given simulation. With the simulator data the whole constitution of the layers as well as their relations with adjacent ones and couplings between Phenomenon objects are built by the MPhyScas engine. After this step, simulation data are supplied through another file and objects in each layer (already built by MPhyScas) will be filled with data dependent on the specific simulation to be performed (constitutive material parameters, geometry data, and so on). Thus, for a given simulator many different simulations may be performed without having to rebuild or redefine it (the simulator).

The main program (”driver”) has the following structure, showed in Figure 6, which shows some methods of the MPhyScas engine employed in the processes described above.

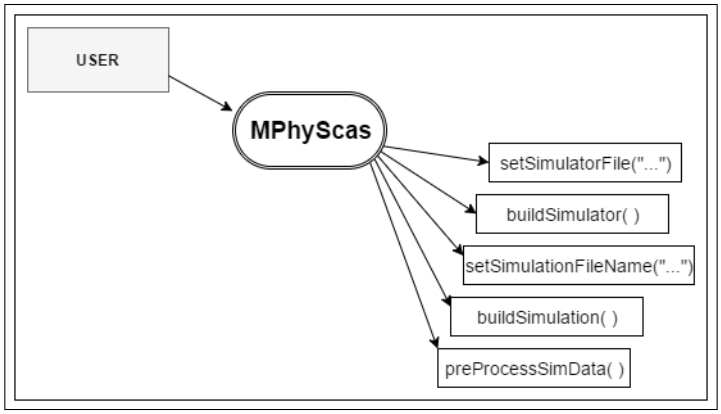


Figure 6: Simulator and Simulation Construction

4.1 System Data Storage

SystData is a structure with the function of storing pieces of data such that they can be shared by objects from the four simulator layers. Any object of any layer can expose and/or request data, and an exposed piece of data should be required by at least one object from other layer.

Required pieces of data should have been exposed by objects from any upper layer or the immediately lower layer. SystData does not allow for exchanging of information between objects belonging to the same layer.

Figure 7 shows the way in which exposed and required data can be share. It can be seen that a exposed data by given Phenomenon may only be required by the next higher layer (Group level). Exposed data by a given Group can be requested by all of its lower layers and the immediately upper one (Group). Exposed data by a given Block may be required by all of its lower layers (Group and Phenomenon) and the next higher layer (Kernel). Finally, data exposed by a given kernel may be required by all lower layers (Block, Group and Phenomenon).

4.2 Preprocessing of simulation data

The preprocessing is divided into two parts, the simulator building and the simulation building. The four layers are formed during the simulator building based on data provided by an XML file. These data file has a well defined structure that must be understood and properly built. In what follows a very short description of the needed data for the preprocessing of MPhyScas is presented, just in order to have an idea on how MPhyScas work in the building of simulators and simulations.

1. Kernel: Only one set of data for Kernel
   (a) System data:
      i. exposed: is a data structure that contains the definitions exposed by the Kernel;
      ii. required: is a data structure that contains the data definitions required by the Kernel and that should be exposed by the Block layer.

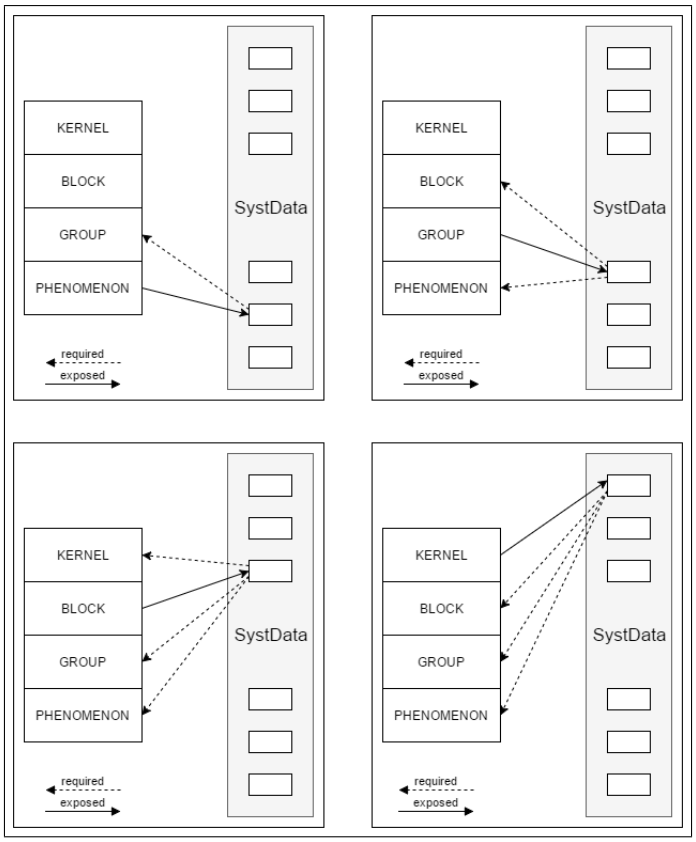


Figure 7: System Data

(b) List of Blocks *ids*: Maps the Block object definition with its identification code;

(c) Kernel Algorithm: there are only one algorithm in this layer;

i. Kernel algorithm parameters.

2. Block: A set of data for each Block

(a) System Data:

i. exposed: is a data structure that contains the definitions of the data exposed by the Block;

ii. required: is a data structure that contains data definitions required by this Block and that are exposed by the Kernel or any Group;

(b) For each algorithm from this Block:

i. *ID* and definition of algorithm

ii. parameters of the algorithm

iii. Ordered list of *ids* of Groups referenced by this algorithm and its respective GroupTasks' *ids*;

(c) List of *id's* of Global States from Block;

3. Group: A set of data for each Group

(a) System Data:

i. exposed: is a data structure that contains the definitions of exposed data by the Group;

ii. required: is a data structure that contains the definitions of the data required by this Group that are exposed by the Kernel, Blocks or some Phenomenon;

(b) gPhen: Group representative in the level of Phenomenon;

i. Solvers: List of linear solvers for this Group;

ii. List of the *id's* of all Phenomenon objects in this Group;

iii. List of execCodes and respective WeakForm objects for gPhen;

(c) Relationship between Global States (Group level) and Vector Fields (Phenomenon level)

(d) Group Tasks: ordered set of specific tasks to be performed by Phenomenon objects

4. Phenomenon: A set of data for each Phenomenon

(a) System Data:

i. exposed: is a data structure that contains the definitions of data exposed by the this Phenomenon;

ii. required: is a data structure that contains the definitions of the data required by this Phenomenon;

(b) Phenomenon Parameter: parameter definitions and settings for Phenomenon;

(c) Spatial and Structural Dimension: These dimensions refer to the space in which the geometric mesh is embedded, as well as the dimension of its manifold;

(d) Vector Fields: They are defined in a Phenomenon mesh and can be scalar, vector or matrix;

(e) Phenomenon Mesh: defines the distribution of order of approximation for vector fields with respect to a geometric mesh;

(f) Geometric Mesh: It is a mesh data structure and depends on the dimension of the mesh;

(g) MasterGeomMesh: It is the mesh in which every other ones are based on;

(h) Geometric mesh generator;

(i) Generator for each phenomenon mesh;

(j) Integration method;

(k) Task set: set of execCodes and respective WeakForm objects

The second part of preprocessing, relates to the construction of the simulation, as well as the geometry and geometric and phenomenon meshes generation. The pieces of data required for the simulation are inserted through an XML file, where the values of the parameters, boundary conditions, among others, are inserted. Below there is an outline:

1. Geometry: Spatial and structural dimension, besides the file (XML) that should contain the geometric data;
2. Kernel:
   (a) Kernel algorithm
      i. Insertion of the values of the parameters existing in the Kernel algorithm.
3. Block:
   (a) Block Algorithm
      i. Insertion of the values of the parameters existing in each Block algorithm.
4. Group:
   (a) gPhen - there is only one, for each Group object
      i. Configure solvers and WeakForm objects for gPhen.
5. Phenomenon:
   (a) Insertion of parameters values;
   (b) data for the building of the geometry graph;
   (c) data for geometry sharing definition;
   (d) data for geometric meshes generation;
   (e) data for phenomenon meshes generation;
   (f) Set of tasks: insertion of data for WeakForm objects.

The example provided in this article is an elastoplastic problem with diffusive coupling (pEPAD). The mathematical formulation is used to construct a solution algorithm and is shown in Section 6, see Chaboche (1990) and Duda et al., (2006) for further information. The most relevant task for the computational implementation of this problem is the mapping procedures from the solution algorithm into the components and respective data for the four hierarchical layers of MPhyScas.

Firstly, the algorithm execution lines are grouped hierarchically. Then, the levels (Kernel, Block, Group and Phenomenon) in which these sets of activity will be placed are defined. Thirdly, we define the Block's, Group's and Phenomenon's objects with their respective sets of activity. So, the Kernel algorithm, the algorithms for each Block, the groupTasks for each Group and ExecCodes and weakforms for each Phenomenon are delineated.

Each set of activity running at a certain level can now be performed with just a command from its immediately higher layer. An example of this is shown in Algorithm 6, where the Kernel requests its Block_0 to execute Algorithm_X. Thus, the pEPAD computational implementation becomes tractable, because, in addition to a structural guide, we end up with simpler software components to be programmed. This simplicity, however, comes with the counterpart of the increasing in the number of components (objects) as one deepens down in the hierarchy of layers.

Algorithm 6 Example: Kernel request to Block_0 execute your Algorithm_X

Kernel
$t = 0$
Block_0 - Algorithm_3
Group_0 - (Elastic) - phen_0 - gPhen
GroupTask— GT_0
$\mathbf{U}_e^{K+1} = \mathbf{U}_e$
$\mathbf{K}_e = \mathbf{K}_e^{aux}$
Group_0 - (Elastic) - phen_1 - Phenomenon
GroupTask— GT_1
$\mathbf{F}_e = \int_e \Psi_j \cdot \mathbf{b} d\Omega$
Elastic Boundary Conditions
Group_0 - (Elastic) - phen_0 - gPhen
GroupTask— GT_2
Pass $\mathbf{K}_e$ to definitive
Group_1 gPhen
GroupTask— GT_0
$\mathbf{U}_d^K = \mathbf{U}_d^n$
$\mathbf{U}_d^K = \frac{2}{\Delta t}\mathbf{U}_d^K$
$\mathbf{U}_d^K = \mathbf{U}_d^K + \dot{\mathbf{U}}_d^n$
$\mathbf{F}_d^{aux} = \mathbf{M} \cdot \mathbf{U}_d^K$
Group_1 Phenomenon
GroupTask— GT_1
$\mathbf{F}_d^{aux} += \mathbf{F}_d^n$
Boundary Conditions

As it could be seen, the execution of the Kernel algorithm starts a sequence of commands, some of which send standardized requirements to the lower layer (Block Layer). Each requirement demanded to an object may need, among other activities, the execution of tasks from objects in the immediately lower layer. In this way, a command emanating from the Kernel algorithm may go down branching into several tasks involving all layers in the simulator. After completing the command sequence in the Kernel Algorithm (Algorithm7), the pEPAD results can finally be analysed.

Algorithm 7 Elastoplastic-Diffusive Problem - Kernel Algorithm

1: $t = 0$
2: Block_0 **execute** Algorithm $A0$
`Allocate memory, Initialize Vector Fields and Time`
3: Block_0 **execute** Algorithm $A1$
`Initial Elastoplastic Calculations`
4: $\Delta t = \Delta t_b$
5: Block_0 **execute** Algorithm $A2$
`Initial Diffusive Calculations`
6: Block_1 **execute** Algorithm $A0$
`Record time, vector fields, mesh`
7: $t = \Delta t$
8: **while** $(t \leq interval)$ **do**
9: Block_0 **execute** Algorithm $A3$
`Compute partial Elastoplastic solution`
10: Block_0 **execute** Algorithm $A4$
`Compute partial Diffusive solution`
11: Block_1 **execute** Algorithm $A1$
`Treat solutions and record data`
12: $\Delta t = \Delta t_b$
13: $t+ = \Delta t$
14: **end while**

pEPAD's solution flowchart is shown in Figure 8, where one can observe the separation between the mechanical and diffusive problems. Note that the diffusive problem is solved only after the solution of the mechanical problem is accomplished.

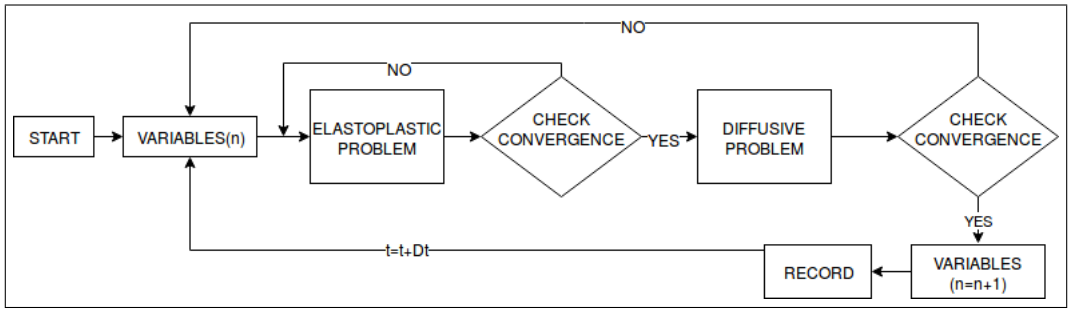
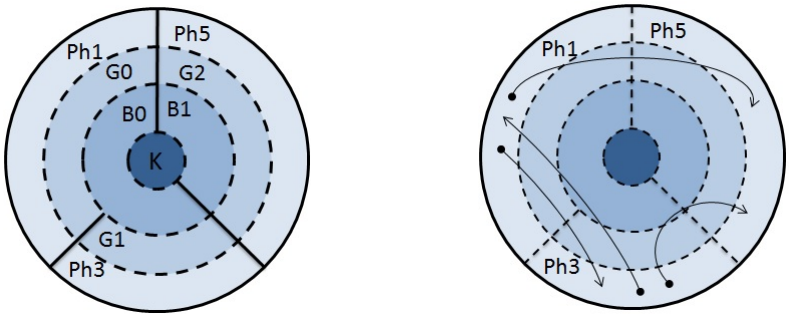


Figure 8: Flowchart

5 SIMULATOR AND SIMULATION

From what was explained above, it is possible to conclude that simulators built by MPhyScas consist of one Kernel, some Block objects, some Group objects for each Block and some Phenomena objects for each Group. Figure 9 presents the configuration choice made for solving the pEPAD (Cruz, 2012) that consists of two Blocks - one is responsible for main processing

of data ($B0$) and the other for post-processing ($B1$) (treating computed results for visualization and recording). Block $B0$ has two Groups, one is related to the elastoplastic behavior ($G0$) and the other to the diffusion problem ($G1$). The amount of related objects in each layer is at the user's discretion, in this case we have chosen one of the simplest, where each object activity can be easily identified by its respective counterpart in the solution algorithm.



(a) Problem built on MPhySCas

(b) Coupling Between Phenomena

Figure 9: Chosen setting

Coupled pieces of data - such as vector fields - are those that belong to a certain phenomenon object and is needed by another one in order to execute some task. They are identified from the solution algorithm and the coupling relationship is defined at the time of simulator building, see Figure 9(b).

The definition of coupled data is straightforward in MPhyScas and it only needs the identification of the coupled phenomenon object and the definition of the respective coupled data to be given to the requiring phenomenon object.

6 CASE STUDY

In this section we present the equations used to model the pEPAD problem; the solution algorithm and its decomposition and mapping into MPhyScas language of patterns leading to the definition of a simulator. Finally, we present the simulation data to be used and depict the results obtained by the simulator.

We start with the equilibrium equation that relates the stress tensor ($\mathbf{S}$) with the body force vector ($\mathbf{b}$). State equations are represented by the stress, the kinematic (Y) and the isotropic ($\mathbf{X}$) hardening equations. In addition to the normality rule of associative plasticity and the laws of evolution and conservation of solute mass (c), we also included the evolution of the cumulative plastic strain (p) and the Kuhn-Tucker and the consistency conditions.

Equilibrium equation:

$$div\left(\mathbf{S}\right) = -\mathbf{b} \tag{5}$$

State equation:

$$\mathbf{S} = \underbrace{\lambda tr\left(\mathbf{E}\right)\mathbf{I} + 2\mu\mathbf{E}}_{\mathbf{S}_{total}} - \underbrace{2e\left(\lambda+\mu\right)\left(c-c_0\right)\mathbf{I}}_{\mathbf{S}_{chemistry}} - \underbrace{2\mu\mathbf{E}^p}_{\mathbf{S}_{plastic}} \tag{6}$$

$$\mathbf{X} = a\beta \qquad Y = r + \sigma_y \qquad r = b\left(1 - e^{-dp}\right) \tag{7}$$

Normality law:

$$\mathbf{N} = \frac{\partial f}{\partial \mathbf{S}} \tag{8}$$

Evolution law:

$$\dot{\beta} = \dot{\mathbf{E}}^p - \frac{\varphi}{a}\beta\dot{p} \qquad\qquad \dot{\mathbf{E}}^p = \dot{p}\mathbf{N} \tag{9}$$

Conservation of solute mass:

$$\dot{c} = -\nabla \cdot \mathbf{J} \qquad\qquad \mathbf{J} = -D\left(\nabla c - \underbrace{\frac{c}{\alpha}\nabla\left(tr(\mathbf{S})\right)}_{coupled}\right) \tag{10}$$

Yield function:

$$f(\mathbf{S}, \mathbf{X}, Y) = \|(\mathbf{S}_{dev} - \mathbf{X})\| - Y = \sqrt{\frac{3}{2}(\mathbf{S}_{dev} - \mathbf{X}) : (\mathbf{S}_{dev} - \mathbf{X})} - Y \tag{11}$$

Kuhn-Tucker conditions:

$$f\dot{p} = 0 \begin{cases} \dot{p} > 0 \quad \text{e} \quad f = 0 \\ \dot{p} = 0 \quad \text{e} \quad f < 0 \end{cases} \tag{12}$$

Consistency condition:

$$f\left(d\mathbf{S}_{dev}, d\mathbf{X}, dY\right) = \frac{\partial f}{\partial \mathbf{S}_{dev}} : d\mathbf{S}_{dev} + \frac{\partial f}{\partial \mathbf{X}} : d\mathbf{X} + \frac{\partial f}{\partial Y} \cdot dY = 0 \tag{13}$$

where λ and μ are the Lam parameters; $\mathbf{E}$ and $\mathbf{E}^p$ are the elastic and plastic strain tensor; c_0 is the initial solute concentration; σ_y is the yield stress; $\mathbf{N}$ is the normal to the yield surface; f is the yield function; $\mathbf{J}$ is the solute flux; D is the diffusive parameter; $\mathbf{S}_{dev}$ is the deviatoric stress; e, a, b, d, φ, α are other constitutive parameters.

An algorithm (Cruz, 2012) was designed based on the set of equations above, whose flowchart is presented in Figure 8. The mathematical problem described above is a model for the diffusion of a solute into a ferrous material that is subjected to mechanical forces.The material behaviour combines elastoplasticity with kinematic and isotropic hardening.

Now we present in a quite simple way how the MPhyScas simulator was defined. The Kernel algorithm was strategically built as follows:
Algorithm 7 shows the Kernel algorithm as it was implemented in MPhyScas. Note that in this hierarchical level there are several demands to the Block layer among other activities like time management and loop.

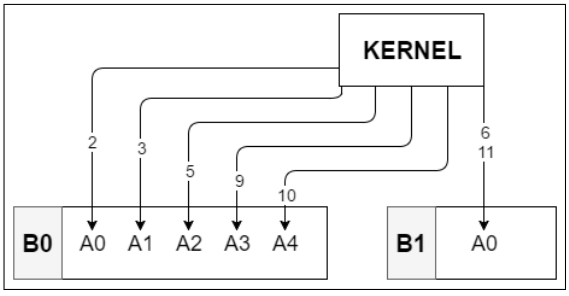


Figure 10: Shows the interaction between Kernel algorithm and Block algorithms

Figure 10 illustrates the interaction between the Kernel algorithm and Block objects, including the definition of what algorithm should be run at each request, which must be consistent with Algorithm 7.

As it was shown before, the execution of any Block algorithm is done only by a request from the Kernel Algorithm. A Block algorithm is a piece of software that implements activities that articulates, among other managing tasks, several demands to be provided by a Group object. Algorithm_8 shows how algorithm $A3$ from Block ($B0$) was implemented.

Algorithm 8 Level Block - $A3$ Algorithm - pEPAD

1: Group_0 **execute** GroupTask $GT3$

```
Prepare elastoplastic variables to solve pEPAD
```

2: Group_1 **execute** GroupTask $GT2$

```
Prepare diffusive variables to solve the pEPAD
```

Figure 11 depicts the relationship between algorithm $A3$ from Block $B0$ and Group objects. Algorithm 8 shows the order in which tasks must be performed.

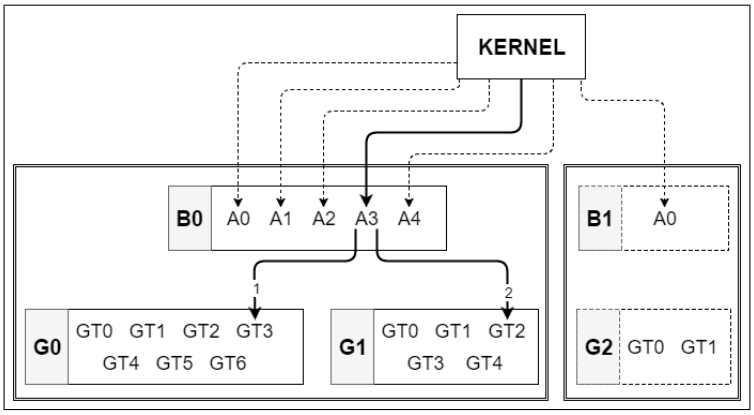


Figure 11: Shows the interaction between a Block algorithm and Group objects

Going deeper in the hierarchical layers, each of the GroupTasks ask for standardized procedures to their Phenomenon objects. Algorithm in 9 shows the way in which Grouptask $GT0$ from Group $G0$ was implemented in pEPAD.

Algorithm 9 Level Group - $GT0$ GroupTask - pEPAD

1: gPhen_0 **execute** execCode $xC0$

```
Compute sizes; Allocate memory for global states
```

2: phen_1 **execute** execCode $xC0$

```
Initialize Vector Fields and Time Step (phenomenon)
```

3: gPhen_0 **execute** execCode $xC1$

```
Initialize Time Step (group)
```

Figure 12 shows how a GroupTask demands activities to be performed by Phenomenon objects. Algorithm 9 shows the order in which tasks must be executed.

From Figure 12, noted that the GT0 GroupTask belong of G0 Group, demands two executions to the gPhen0 and another one to the Phen1, following the numbered showed in figure and

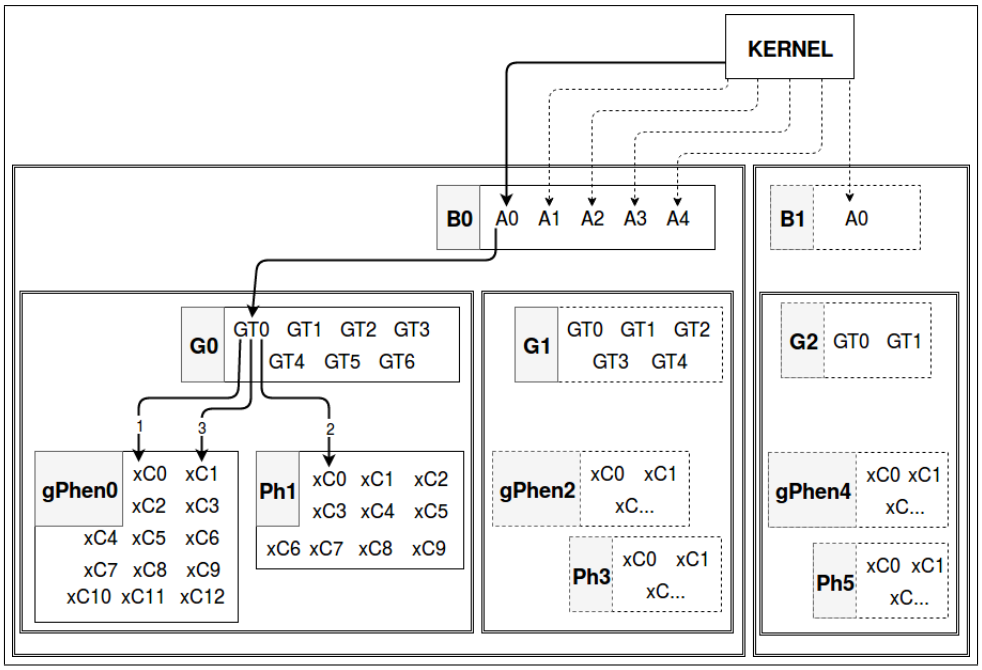


Figure 12: Shows the interaction between a groupTask object and Phenomena objects

presented in Algorithm 9. These runs are a list of codes that represent the WeakForms to be used following an order presented by the user (during the simulator build). To the pEPAD are showed bellow the WeakForms that are used for each one of execCodes requested by GT0.

1. gPhen_0
   (a) - xC0 -
      i. WeakForm WF_0, to compute size of vector fields
      ii. WeakForm WF_1, to allocate the vector fields
   (b) - xC1 -
      i. WeakForm WF_2, to compute the time step of Group
2. Phen_1
   (a) - xC0 -
      i. WeakForm WF_3, to initialize the vector fields
      ii. WeakForm WF_4, to compute the time step of Phenomenon

Next we provide some of the simulation data, particularly the geometry data. The geometry of the model problem is a plate with a circular hole in the middle and a notch on the lower side. This plate is restricted in the tangential direction on the left side and on lower left edge. It is subjected to a tangential force on the right edge. Figure 13 shows this geometry and its geometric mesh.

Initially, the plate has a uniform solute distribution and the application of mechanical loads along the time forces this solute to move through the crystal lattice of the material.

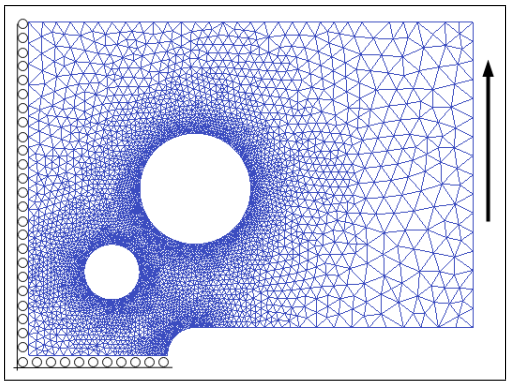
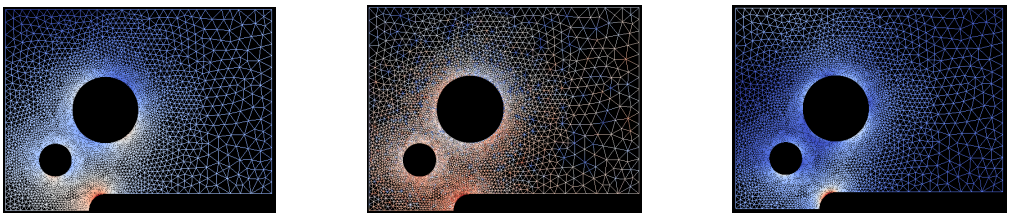


Figure 13: Configuration used to problems

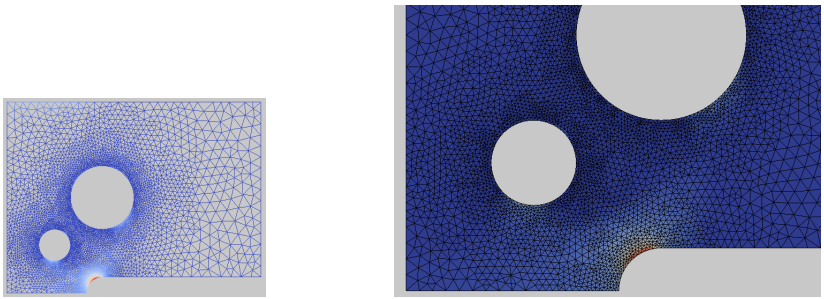
The results obtained by the simulation are shown in Figures 14(a), 14(b) and 14(c) depicts the solute migration to the regions where the hydrostatic stress is greater. Therefore, the regions where the hydrostatic stress is smaller ended up with lower solute concentrations, because there is neither solute creation nor solute destruction nor escaping of solute through the boundary. Through this case study, we notice the importance of simulations. It is possible to observe that the migration of the solute make them even more critical the regions where stresses are already high.



(a) Solute Concentration (b) Hydrostatic Stress (c) Von Mises Stress

Figure 14: Elastoplastic-Diffusive Problem

The cumulative plastic strain is shown in Figure 15. Only in the detail, Figure 15(a), it is possible to observe the non-zero cumulative plastic strain distribution on plate. It is, then, possible to note that the cumulative plastic strain initiated on the boundary and propagated inwards.



(a) Cumulative plastic strain (b) Details of Cumulative Plastic Strain

Figure 15: Elastoplastic-Diffusive Problem

Based on the results the following Table 1 is built:

MPhyScas is a computational framework made to help in the design and implementation of simulators for coupled problems using the finite element method. The main purposes are related to increasing the software reusability, maintainability and adaptability, together with allowing for multidisciplinary cooperative work and knowledge management. All of those properties may enable modifications of a solution algorithm within a minimum time and resource cost. For instance, problems of great complexity can be understood as a number of less complex sub-

Table 1: Maximum and minimum values at the end of the simulation

	Elastoplastic-Diffusive	
	Minimum value	Maximum value
Solute concentration	$2.99e^{-5}$	$3.01e^{-5}$
Von Mises Stress (Pa)	$1.89e^{6}$	$6.06e^{8}$
Hydrostatic Stress (Pa)	$-8.00e^{8}$	$6.90e^{8}$
Cumulative plastic strain	$0.0e^{0}$	$1.67e^{-3}$

problems whose solutions may already have been programmed (with components recorded in the repository), leaving the user to understand its operation and to adapt it to the new problem environment, with the correspondent decrease in the amount of time spent in the simulator development.

REFERENCES

Santos, F. C. G., Barbosa, J. M. A., Bezerra, J. M., Brito, E. R. R. J., & Santos, I. H. F., 2007. Towards the Automatic Development of Simulators for Multi-physics Problems. *International Journal of Modeling and Simulation for the Petroleum Industry,* vol. 1, n 1.

Ross, C. T. F., 1996. Finite element programs in structural engineering and continuum mechanics. *Albion Engineering Science Series.*

Simo, J. D., & Mieche, C., 1992. Associative coupled thermoplasticity at finite strains: Formulation, numerical analysis and implementation. *Computer Methods in Applied Mechanics and Engineering*, Elsevier Science Ltd, n. 98, p. 41-104.

Shaidunov, V. V.,& Marchuk, G. I. 1983. Difference methods and their extrapolations. *Application of Mathematics*.

Santos, F., Lencastre, M., & Vieira, M., 2003. Workflow for simulators based on finite element method. *Proceedings of the International Conference on Computational Science (ICCS)*, Melbourn, Australia and Saint Petersburg, Russia.

Lencastre, M., 2004. *Conceptualisation of an Environment for the Development of FEM Simulators.* Thesis (Computer Science) - Universidade Federal de Pernambuco, Recife, Pernambuco.

Chaboche, J. L., 1990. *Mechanics of solids materials*. Cambridge University Press.

Duda, F. P., Guimarães, L. J., Souza, A. C., & Barbosa, J. M., 2006. On the Modeling of Deformation-Diffusion-Damage Coupling in Elastic Solids. *III European Conference on Computational Mechanics. Solids, Structures an Coupled Problems in Engineering*. Lisbon-Portugal.

Cruz, F. A., 2012. *Desenvolvimento de um Simulador para Sistemas com Acoplamento Elasto-plastico e Difusivo.* Dissertação em Engenharia Mecânica. Universidade Federal de Pernambuco. Recife - PE.