Pós-Graduação em Ciência da Computação

"Conceptualisation of an Environment for the Development of Simulators based on the Finite Element Method"

Por

Maria Lencastre P. Menezes e Cruz

Tese de Doutorado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, FEVEREIRO/2004

Universidade Federal de Pernambuco

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MARIA LENCASTRE PINHEIRO DE MENEZES E CRUZ

"CONCEPTUALISATION OF AN ENVIRONMENT FOR THE DEVELOPMENT OF SIMULATORS BASED ON THE FINITE ELEMENT METHOD"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INNFROMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIA DA COMPUTAÇÃO

ORIENTADOR: Prof. Jaelson Brelaz Castro
CO- ORIENTADORES: Prof. Felix C. Guimarães Santos
Prof. João Araújo

RECIFE, FEVEREIRO/2004

Cruz, Maria Lencastre Pinheiro de Menezes e
Conceptualisation of an environment for the development of simulators based on the finite element method / Maria Lencastre Pinheiro de Menezes e Cruz. – Recife : O Autor, 2004.
xi, 181 folhas : il., fig., tab.

Tese (doutorado) - Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2004.

Inclui bibliografia e apêndices.

1. Engenharia de software (Padrões e *Framework*). 2. Modelagem e simulação – Método do elemento finito). 3. Engenharia dos requisitos (*Problem frames*). 4. Domínio do conhecimento (Método do elemento finito). I. Título.

004.414.38	CDU (2.ed.)	UFPE
003.3	CDD (21.ed.)	BC2004-208

To my parents
and
to my daughters Isabel and Beatriz

“Os donos do futuro são aqueles que conhecem o poder da cooperação, trabalham em equipe, armam seus times antes de realizar um projeto e lutam até alcançar os seus sonhos”

(Roberto Shinyashiki)

Abstract

In this work we address the conceptualisation of a Simulation Environment for the development of multi-physic simulators based on the Finite Element Method (FEM). Simulators are economical means of understanding and evaluating the performance of abstract and real-world systems. Our simulation perspective is the class of simulations for phenomena represented by a set of functions distributed in space and possibly in time, whose behaviour is based on the FEM. The importance of these simulators has to do with the effectiveness of the FEM, a general-purpose numerical method, which can easily be developed to analyse and solve various kinds of problems frequently found in human daily life, and in its power to provide accuracy and reliability in the solution of partial differential equations. FEM Simulations consider systems of possibly millions of algebraic equations, numerical integrations, mesh generations, matrix and vector manipulations, solutions of linear and non-linear systems, and so on. These features undoubtly justify the development of a specific computational environment. This work emphasizes the adaptation of software engineering practices and methodologies for organizing and reusing the specific domain of simulators formulated using the FEM. The work defines an environment and its architecture for the development of simulators. It also proposes some specific patterns for solving relevant problems of our domain of knowledge, and describes their application through a case study.

Key words*: Modelling and Simulation, Framework, Patterns, Finite Element Method, Problem Frames, Requirements Analysis and Knowledge Domain.*

Resumo

Este trabalho tem como principal objetivo a conceitualização de um ambiente de simulação para o desenvolvimento de simuladores multi-física, baseados no Método de Elemento Finito (MEF). Os simuladores são meios econômicos de compreender e de avaliar o desempenho de sistemas abstratos e do mundo real. Nossa perspectiva de simulação é a classe das simulações para os fenômenos representados por um conjunto de funções distribuídas no espaço e possivelmente no tempo, cujo comportamento é baseado no MEF. A importância destes simuladores tem a ver com a eficácia do MEF, um método numérico de propósito geral, que é facilmente desenvolvido para análise e resolução de vários tipos de problemas encontrados freqüentemente na vida diária, e devido ao seu poder de fornecer exatidão e confiabilidade na solução de equações diferenciais parciais. As simulações baseadas no MEF consideram sistemas possivelmente de milhões de equações algébricas, de integrações numéricas, de geração de malhas, manipulações de matrizes e vetores, solução de sistemas lineares e não lineares, e assim por diante. Estas características justificam, sem dúvida, o desenvolvimento de um ambiente computacional específico. O trabalho realizado enfatiza a adaptação de práticas e de metodologias da tecnologia de programação para organizar e permitir o reuso do domínio específico dos simuladores formulados usando o MEF. O trabalho define um ambiente e a sua arquitetura para o desenvolvimento de simuladores. O trabalho também propõe e define padrões de modelo e de projeto específicos para a solução de problemas relevantes no domínio do conhecimento sendo tratado. A aplicação destes padrões foi realizada através de um estudo de caso.

***Palavras chave**: Modelagem e Simulação, Framework, Padrões, Método do Elemento Finito, Problem Frames, Análise de Requisitos e Domínio do Conhecimento.*

Summary

List of Figures

List of Tables

Chapter

1

Introduction

This chapter presents the main motivation for this work; describes the importance of simulators, the impact of computational mechanics and the use of the Finite Element Method on several contexts. Then, it details the contribution of this work, which is the conceptualisation of a Simulation Environment for the specification and control of simulation models for coupled multi-physics phenomena. The target users of this proposed environment are the developers of numerical programs, academics and researchers. Finally, the organization of this work is presented.

1.1 Motivation

We can guess that in the future computers will properly fade into the background in at least two ways – into appliances that play some part in our lives, and into simulations that present us with intriguing environments in which we interact. Such environments range from the fanciful worlds of science fiction, interactive games and animation to engineered simulations of complex systems that exist only in the mind; and to environments in which individuals can learn and groups can be trained as teams [KWD99].

In this work we will focus our attention on simulation, which is a highly relevant method for solving many real world problems. Simulation is used to describe and analyse the behaviour of a system, ask what-if questions about real systems and aid in their design. Both real and conceptual systems can be modelled and simulated. Simulators provide an economical means of understanding and evaluating the performance of both abstract and real-world systems. Unfortunately, the design and implementation of simulators is almost as complex as the systems being simulated. Therefore, in order to be efficient, simulators must be able to adapt to an ever-increasing system complexity.

Modelling is an important and perhaps the primary tool for studying the behaviour of large complex systems [BAJ98]. In physical sciences, models are usually developed based on theoretical laws and principles. These models may be scaled up to physical objects (iconic models), mathematical equations and relations (abstract models), or graphical representations (visual models). The usefulness of models has been demonstrated in describing, designing, and analysing systems. In [BAJ98], one can find some principles of modelling:

- Conceptualising a model requires system knowledge, engineering judgment and model-building tools.
- The secret of being a good modeller is the ability to remodel.
- The modelling process is evolutionary. The resulting correspondence between the model and the system not only establishes the model as a tool for problem solving but also provides system familiarity for the modellers and act as a training vehicle for future users.

Developing a validated simulation model involves three basic entities: real/conceptual world system under consideration, a theoretical model of the system, and a computer based representation of the model. The activity of deriving the theoretical model from the real world system can be referred to as simulation modelling, and the activity whereby the computer based representations derived from the model can be referred to as simulation programming [GV95] (see Figure 1-1).

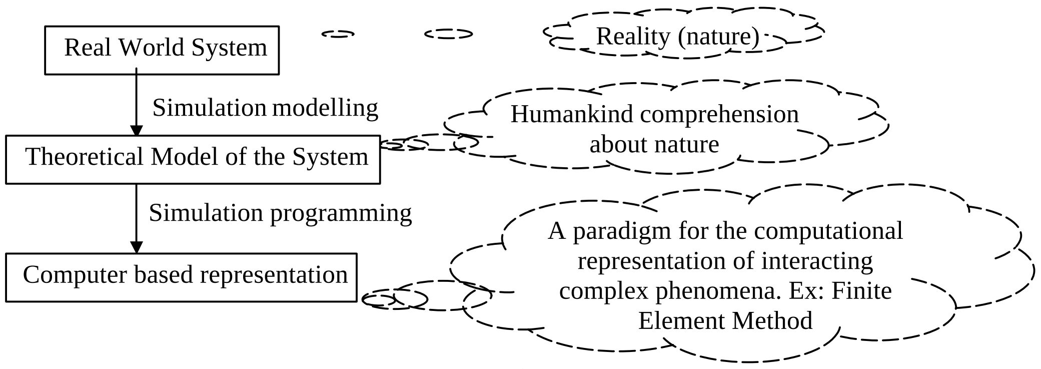

Real World System
Simulation modelling
Theoretical Model of the System
Simulation programming
Computer based representation
Reality (nature)
Humankind comprehension about nature
A paradigm for the computational representation of interacting complex phenomena. Ex: Finite Element Method

Figure 1-1 Basic entities of a simulation model development

Scientists and engineers write most of their own technical software. It is not likely that someone from another field could write a program from what they found in numerical analysis journals, because those papers are very mathematical [GR02]. It could be speculated that numerical analysis specialists would be dispersed among the sciences; so maybe those people are doing the programming. If so the development of numerical analysis software is even more of a concern. This happens, because most software engineering techniques do not emphasize specific development methodologies to support these types of developers. Therefore, numerical analysts are required to go deeper in the computer world, spending time developing proper abstractions and having to deal with problems not of their specific area and interest.

When a designer defines a computational model for a mathematical formalism, he/she has to deal with model complexities used to limit the assessment of their correctness (model verification). Usually, the designer produces restricted documentation about the generated code, and does not provide or use high levels of abstraction. Thus, the experiments with those models can hardly be replicated without access to the entire set of numerical models and solution techniques. Hence, the issue of information reuse goes beyond the support provided for restricted communities of researchers.

This work describes a method for improving the development of simulators in the specific domain of Computational Mechanics. Computational Mechanics has had a profound impact on science and technology over the past three decades. The Computational Mechanics software industry moves several billions of dollars per year. However, in many aspects, it applies software engineering development techniques related to the seventies. The success of Computational Mechanics is due to its effectiveness in solving problems that interest society and in providing deeper understanding of natural phenomena (physical facts like motion and heat transfer) in engineering systems. We can define *Multi-physics* as a qualifier for a set of interacting phenomena, in space and time. These phenomena are usually of a different nature (deformation of solids, heat transfer and electromagnetic fields) and may be defined in different scales of behaviour (macro and micro mechanical behaviour of materials).

Sometimes a multi-physics system is also called a *coupled phenomena* system. As an example of multi-physic analyses, one may consider:

- Analysis of air conditioning (environment thermal comfort): evaluation of temperature distribution, and air movement inside a room.
- Analysis of air resistance, considering the airflow around cars and airplanes in movement.
- Thermal stress analysis: engineers can simulate gradual or rapid temperature changes and predict deflections or stresses occurring in an object, whose resistance to mechanical efforts depends on temperature.

The enormous success of Computational Mechanics resides in its predictive power, making possible the simulation of complex physical events and the further use of these simulations in the design of engineering systems. This is done through the so called computer modelling: the development of discretized versions of the theories of mechanics, which are amenable to digital computation, together with complex processes of manipulating these digital representations to produce abstractions of the way real systems behave [TO95]. The Finite Element Method (FEM) has been frequently used in the field of Computational Mechanics, which has come to rely heavily on this technique. Gradually the FEM is becoming the most popular analysing procedure within various fields of design [VM02].

The FEM is a way of obtaining a numerical approximation of a mathematical theory, which describes physical behaviour. This method is considered a powerful computational technique for the solution of differential and integral equations that arise in various fields of engineering and applied science [COR95]. There is an important feedback in developing simulators using the FEM due to its great applicability in making previsions in several contexts, such as:

- Systems involving *fluids*, which can range from simple fluids like water up to more complex ones such as blood, air or petroleum.
- *Damage evolution mechanisms*, for example, describing damage arising and the development in materials (from airplanes to biological organs).
- *Heat Transfer Systems*, involving conduction, convection and/or radiation.
- *Solid Systems* such as vehicles parts, civil construction, biologic organs.
- *Chemical Reactions:* (i) In the environment: water quality (biologic environment where chemical reactions occur); air quality; pollutant dispersion; (ii) Reactors (industrial reactions).

This work is part of the Plexus[1] project, whose objective is the development of a computational environment to help the design, implementation, validation and verification of a simulation software for coupled phenomena based on FEM [LSA01].

[1] Plexus's catchy name came from the involved com<u>plex</u>ity treated in the project.

The work focuses on the simulation of chemo-thermo-mechanical interactions, which occur inside a given system, and between this system and its surrounding environment. Our case studies are based on applications related to Petroleum (damage evolution in pipeline networks conveying fluids) and Medicine (integration of image acquisition systems and simulation systems) that are parts of current research projects carried out in the Mechanical Engineering Department, UFPE, Brazil.

The simulation perspective, used in this thesis, deals with the class of simulations for phenomena represented by a set of functions distributed in space and possibly in time, which are applied to part of a specific domain of Computational Mechanics - the Simulation of Coupled Multi-physic Systems - using the FEM. This work investigates ways to support flexible techniques to help the construction of these types of simulations.

The method to be developed should be: (i) adaptable to new technologies and trends (such as: intelligent systems, distributed simulation, cooperative work, and open systems), (ii) extensible and flexible to incorporate new knowledge in the form of new models and solution algorithms.

The next section describes the major contributions of this thesis.

1.2 Contribution

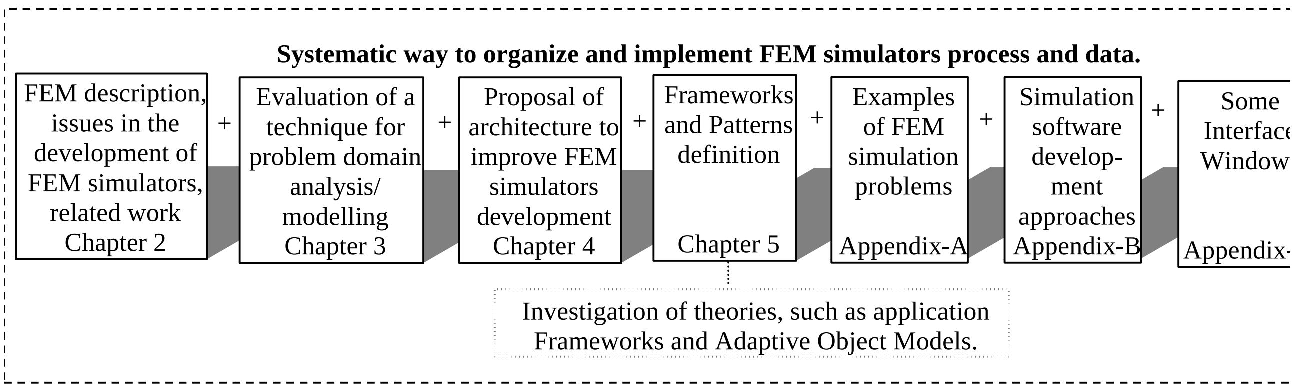
The use of some paradigms and the extension of modern software engineering solutions represent some of the aspects that make possible the achievement of new degrees of attained satisfaction and gains in FEM simulators conception and project development. The important idea of domain-oriented reuse is stressed in this work. In fact, in order to be effective, reuse has to be performed at a high level of abstraction, that is, at the domain level [NEI84, LJ99]. In this work we explore the FEM domain.

The objective of this work is to reduce the gap between software engineering solutions and the way our specific domain of knowledge (the FEM domain) has been conceptualised up to now.

Thus, we propose a Simulation Environment, called Plexus, which takes into account existing expertise and facilitates improvement of software features through use of valuable scientific methods and techniques. By definition FEM is deeply polymorphic, which means that highly versatile FEM based-simulators may be produced. Such generality does not exist in classical analytical methods [COR95, RG00]. Hence, we describe a domain specific proposal to detail the systematic way to define, organize and implement the FEM data and processes.

The main contributions of this thesis are presented in Figure 1-2:

- Exploration and adaptation of some software engineering techniques, such as Problem Frames [JAC01], emphasizing the importance of a good description and analysis of the problem domain, increasing the problem completeness and comprehension, independently of the proposed solution strategies.
- The analysis and indication of domain specific ways for FEM representation and implementation, decreasing the difficulty, time spent and development costs of FEM simulators. This is realized through the definition of:
    - The Plexus Simulation Environment architecture for the domain being considered; focusing on process reuse. It specifies an abstract process for the solution of coupled phenomena simulators, based on pre-defined scenarios. It supports a framework for the construction of semi-complete simulators; which can be further configured for the definition of more specific ones.
    - Specific patterns[2] applied to the Plexus architecture, such as the ones intended to: (i) guide the development of simulators models based on FEM; (ii) define and control simulator process flows, taking into account some specific requirements of the domain being treated, assisting in the design and reuse of programs, (iii) encapsulate simulators low level and complex procedures and relationships, which are very specific to the FEM model of single phenomenon behaviour laws.
- Development of case studies, to validate the proposal.



Systematic way to organize and implement FEM simulators process and data.
FEM description, issues in the development of FEM simulators, related work Chapter 2
+
Evaluation of a technique for problem domain analysis/ modelling Chapter 3
+
Proposal of architecture to improve FEM simulators development Chapter 4
+
Frameworks and Patterns definition Chapter 5
+
Examples of FEM simulation problems Appendix-A
+
Simulation software develop-ment approaches Appendix-B
+
Some Interface Window Appendix-
Investigation of theories, such as application Frameworks and Adaptive Object Models.

Figure 1-2 Simulation Environment Conceptualisation

Our thesis includes:

- The use of object-oriented simulation as a basic paradigm. The FEM mathematical representation already has many properties, which are directly linked to object-oriented concepts [ZL97], such as: abstraction, polymorphism, encapsulation, and modularity.

[2] Patterns describe ideas and perspectives [FAY99]. A pattern is a small collection of atomic units and a description of their relationships. To be relevant, a pattern must express a general recurrent theme that has proven to be useful.

- Adoption of well-known software engineering techniques such as: Frameworks [FJL01], Adaptive Object Models [YJ02], and Workflow [MAN01]. They seem to be relevant and suitable to improve the architectural definition of FEM simulators.
- Definition of a collection of patterns for FEM, to be used to describe abstractions for the conception of simulators, and the main complex steps and solutions for a simulation process definition. This improves the way in which software components can be developed.

This proposal covers from requirements engineering up to the design of a simulator engine, which is part of our specific domain of Computational Mechanics. The results of this work point to simulator architecture, implemented by a specific framework, which supports more flexible and rich computational solutions for the development of FEM simulators. Our focus is on software quality, reuse of models and data, pattern definition, and process improvement.

We can summarize and distinguish the different and interdisciplinary areas involved in our work:
a) Computer Science, specifically in the following sub-areas:

- Simulation Modelling: to study and model simulations, which involve a set of dependent and distributed functions (that develop over time or not).
- Software engineering: explored for a specific Domain of Knowledge. The conclusions obtained might also help other domains of knowledge that have similar characteristics, increasing the relevance of the achieved results:
    - Development and evaluation of a method for modelling and analysing requirements in a complex domain of knowledge [LCS03].
    - Development of a specific architecture (framework) that focus on quality attributes such as process reuse and system flexibility [LSR02b].
    - Proposal of new patterns, for the specific domain of computational mechanics considered [LSR02a], which could be further applied to similar domains. This includes: the definition of abstract processes and structures [LSA02b], giving support to specific simulator components development; workflow analysis in our specific domain [LSV03a].

b) Computational Mechanics, where the final users will gain more powerful and specific abstractions and techniques for the development of FEM simulators.

The next section describes the organization of this thesis.

1.3 Organization of this Work

This chapter presents the main motivation for this work, which is related to the power of simulators in multi-physics systems, within the field of Computational Mechanics. The thesis contribution was also presented, giving a brief description of the involved solutions and research areas. The remainder of the work is organized in the following way.

Chapter 2 provides a background for simulations of coupled multi-physics phenomena using the FEM. It details the processes and concepts involved. This leads to the identification of some important issues that allowed for a better understanding of the design of finite element programs. Generic requirements for the development of FEM simulators are presented. Finally, related work illustrating what is currently being developed is shown.

Chapter 3 investigates a suitable description technique for the specific domain of mechanics, that is, the FEM simulators development for coupled multi-physics phenomena. It considers the Problem Frames technique to improve the description of the Plexus Simulation Environment. The main characteristic of Problem Frames technique is that it separates the domains into problem and solution domains. The problem is not at the computational interface – it is routed inside the world, further away from the computer.

Chapter 4 presents the Plexus Simulation Environment architecture for supporting the simulation of coupled phenomena, based on FEM solutions. The chapter presents architectural components and the interaction between them and their functionalities. This architecture gives a clear perspective of the whole environment and the control required for its development, and tries to reflect some system requirements and quality attributes such as reuse, modularity and flexibility.

Chapter 5 describes some architectural abstractions, such as frameworks and domain specific patterns, whose purposes are the specification of domain solutions (features and structural characteristics) to be used in the Plexus Simulation Environment. The patterns include: a Computational Phenomena Pattern (a pattern which standardizes the complex model of a phenomenon and makes intuitive and easier the representation of data sharing and dependence between different phenomena); a FEM-Simulator Skeleton Pattern (a pattern for modelling FEM simulators based on algorithm skeletons for coupled phenomena), a GIG-Pattern (a pattern based on a Generic Interface Graph for process control). A case study is presented and applied during the description of these patterns.

Chapter 6 summarizes the objectives and contributions of this work in the conceptualisation of a Simulation Environment, it describes some identified limitations, future activities and contains some final remarks.

Appendix A presents three different examples of problems involving coupled multi-physics phenomena. First we give a description of a simulator that can be used to solve two of the given problems (example 1 and 2). In addition, in the first example (number

1) we present extra details about exact mathematical models, the differential-algebraic system of equations, and the global algorithm for the problem. These details can help on a more complete understanding of FEM patterns

Appendix B presents several ways to develop simulation systems, including some examples of general-purpose languages, simulation languages and simulation environments (specific and general purpose).

***Appendix* C** gives a brief explanation about Plexus system and also shows its interface.

Chapter

2

Simulation of Coupled Multi-physic Systems using Finite Element Method

This chapter provides a background for simulations of coupled multi-physics phenomena using the FEM. It details the processes and concepts involved. This leads to the identification of some important issues that allows a better understanding of the design of finite element programs. Generic requirements for the development of FEM simulators are presented. Finally, related work illustrating what is currently being developed is shown.

2.1. Introduction

Real physical systems are highly complex encompassing several factors. The modelling of such systems chooses the most relevant variables to be represented, reducing their complexity. Sometimes, the translation of the "domain" of the problem into a different one may be convenient. This is the case whenever one wants to model structures of complex (and irregular) shapes. The FEM simplifies the modelling by splitting the original surface into small regular pieces, which "cover" the whole object to be modelled. This technique may provide the answers to some interesting questions, for an example see Figure 2-1.

A complex structure like a helicopter can be simulated on a computer so that the helicopter's physical properties can be studied to determine how well the design will perform under real world conditions. The computer models permit the design team to examine a wide range of options to detect design flaws long before the prototype stage.

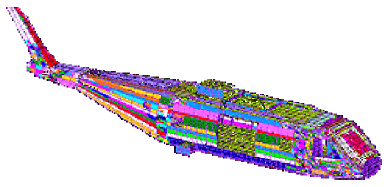


Figure 2-1 Example of an Engineering Problem

The visualization technique, used in conjunction with engineering analysis, tries to provide the most meaningful way for engineers to view both "input" and "output". The complete process is described by a sequential cycle composed of the following steps: build a model, analyse it, view results, review its behaviour, then, change the model and repeat the cycle until a satisfactory result is obtained, see Figure 2-2.

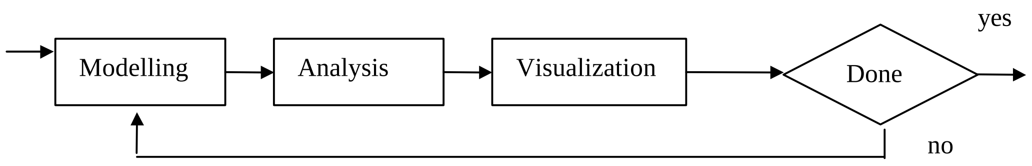


Figure 2-2 Cycle of engineering analysis process

The functions, which take place before and after analysis, are often described as "pre-processing" (it takes real data and generates data in the form accepted by the models) and "post-processing" (that takes analysis output and generates data required by the user in the form accepted by the viewer).

Many models applied to engineering and applied science problems are governed by differential or integral equations. The solutions to these equations would provide an exact solution to the particular model of the problem being studied. However, complexities in the geometry, material properties and in boundary conditions - that are seen in most real world problems - usually mean that an exact solution is not available. For such classes of problems, where the analytical solutions are not available, computational solutions have a great impact on the way the search for solutions occurs and in the way engineering projects are made.

Many typical modern engineering and analysis techniques used in computational solutions are based upon discretizing a problem domain into small pieces or elements. For each element an approximate function is used to represent the behaviour of the element in terms of unknown solution variables. These variables are gathered into large systems of equations and then solved on a computer. The results of this, permits the definition of the desired approximate solution. Engineers try to find convergence in the approximate solution, considering that if errors can be estimated and models can be adapted, then results can be improved. In particular, we are interested in methods, which are used in the development of approximate procedures that can be applied in a general context, inside the limits of the acceptable precision of the engineering problem taking into account reasonable use of time and money.

The FEM has been considered as one of the major methods for finding approximate solutions to systems of coupled partial differential equations. One of the main advantages of the FEM is that it allows the development of computational systems aimed at solving various classes of problems [YB96]. The FEM proposes to solve complex problems modelled as phenomena that occur in continuous media represented as vector fields defined in these media. By continuum we mean a body of matter or simply continuous region of space in which a particular phenomenon is occurring. This can be a piece of metal subjected to a difference of temperature, a region of space under a magnetic field or a fluid subjected to a mechanical load. In any case, we are after the distribution of the field variable resulting from imposed conditions and behaviour laws.

The FEM has been extensively used in the analysis of structural designs for over three decades. This method has become the *de facto* industry standard for solving multi-disciplinary engineering problems that can be described by Integra-algebraic-differential equations. Its influence cuts across several industries by virtue of the applications – solid mechanics (civil, aerospace, automotive, mechanical, biomedical, and electronic), fluid mechanics (geophysics, aerospace, electronic, environmental, hydraulics, biomedical, and chemical), heat transfer (automotive, aerospace, electronic, and chemical), acoustics (automotive, mechanical, and aerospace), and electromagnetism (electronic and aerospace), etc.

The usual method for developing and making the required analysis is to select or derive a mathematical model, appropriate for the physical problem in mind, which can accept as input the geometry, material properties, known restrictions, initial conditions and other pieces of data. Typically the mathematical models produced by this process are systems of partial differential equations, see Figure 2-3.

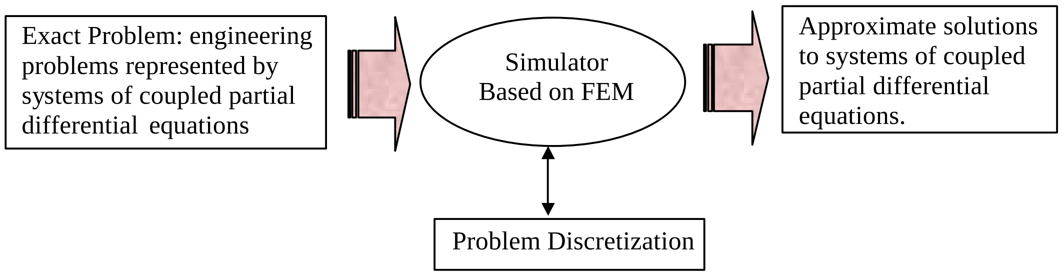

Exact Problem: engineering problems represented by systems of coupled partial differential equations
Simulator Based on FEM
Approximate solutions to systems of coupled partial differential equations.
Problem Discretization

Figure 2-3 Input and Output of a Simulation Based on the FEM

The construction of a geometric model describing the problem geometry is used to create the geometric data needed for the analysis. The same analysis model can conceivably be used to solve a number of states of behaviour (e.g. different loads and initial states) in different geometries.

The FEM solves the problem through an approximation method, where it computes the solution of an algebraic system of equations. The solution to that system is a vector of coefficients of a linear combination of known functions, which is the approximate solution. Thus, the produced outputs are the approximate vector field and desired response variables computed over the geometric domain and based on that approximate solution.

The Exact Problem

The planning of a simulation should consider, from the beginning, the definition of all physical phenomena and respective vector fields. A physical phenomenon can be abstractly defined as a fact or occurrence, which can be described by a certain finite number of pieces of information, which, in turn, have to obey a set of behaviour laws. For example, fluid flows and heat transfer.

An exact problem definition includes:

- Exact Geometry where the phenomenon is defined;

- Vector Field, which denotes a vector-valued function that describes a phenomenon, defined over the exact geometry. In Figure 2-4 there are some examples of vector fields.
- Phenomenon Behaviour Laws is an **exact mathematical formulation** of a phenomenon, comprised of a system of Integra-algebraic-differential equations, which governs the behaviour of the phenomenon vector field. There is one behaviour law for each vector field. Restrictions may be applied to the vector field such as boundary conditions and others. There is an equivalent way of describing the behaviour laws, which is called the exact weak form. The latter is the form used by the FEM.

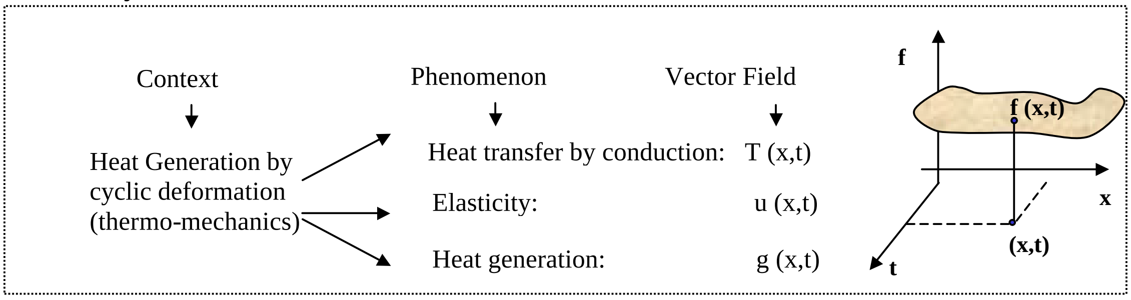


Figure 2-4 Vector Field Examples

Differential equations are, for example, the ones that describe the displacement of a material body and temperature distribution. Figure 2-5 shows an example of a behaviour law, which can be used for one-dimensional linear elasticity and heat transfer. It includes the boundary conditions $u(0) = 0, u(1) = 0$. These laws will be applied later, in this section, for the representation of phenomena occurring on a geometry (a curve), represented in Figure 2-9.

$$\begin{cases} -\dfrac{d^2u}{dx^2} + u = x, 0 < x < 1 \\ u(0) = 0, u(1) = 0 \end{cases}$$

Figure 2-5 Example of a Behaviour Law represented by a Differential Equation

This chapter is organized in the following way. Section 2.2 details the major FEM concepts. Section 2.3 introduces coupled phenomena and provides some application examples. Section 2.4 presents the identification of the major issues and requirements for the development of FEM-based simulators for coupled multi-physics phenomena is. Furthermore, some related works are described in section 2.5. Finally, some considerations are presented in section 2.6.

2.2 FEM Concepts

Classically speaking, almost all the FEM variants have a lot in common, for instance:

- A specific way to describe the behaviour laws of the involved phenomena (the so called exact weak form).
- A geometry where phenomena are defined (exact geometry).
- A discretization of the exact geometry into a collection of simple geometric forms (the geometric mesh). Those simple geometric forms are called finite elements.
- Definition of special functions based on a given mesh (shape functions).
- Use of shape functions for the definition of the basis for two finite dimensional spaces of functions: the discrete trial set and the discrete test space.
- Definition of the approximate solution as a member of the discrete trial space, through a linear combination of the components basis (the coefficients are the unknowns of the problem and may be time dependent or not).
- Definition of the discrete behaviour law (discrete weak form) based on the exact weak form, of the approximate solution (discrete vector field) and on the discrete test space. The discrete weak form may still be modified for reasons related to numerical stability and/or solution method requirements.
- If the problem is not time-dependent, the final result of the discrete behaviour law is either a linear or non-linear system of algebraic equations, whose unknowns are the coefficients used in the definition of the approximate solution.
- If the problem is time-dependent, the final result of the discretization is either a linear or non-linear system of ordinary differential equations (the so called semi-discrete equations), whose unknowns are the time-dependent coefficients of the approximate solution. For time-dependent problems, a scheme for discrete (in steps) time progression is defined in order to compute the state of the problem at each time instant, starting from a given initial state.
- The solution of a system of non-linear/linear algebraic equations.
- Auxiliary methods, which execute specific processes such as: mesh generation, numerical integration at the finite element level; a posterior error estimations for the discretization and models; adaptations of discretization and of models; and solvers for non-linear and linear systems of algebraic equations; etc.

We can summarize by saying that a system of algebraic equations is the result of a process, which involves the discrete sets of shape functions (trial and test), the discrete weak form, the discrete vector fields, the geometric mesh and other data (see Figure 2-6[1]).

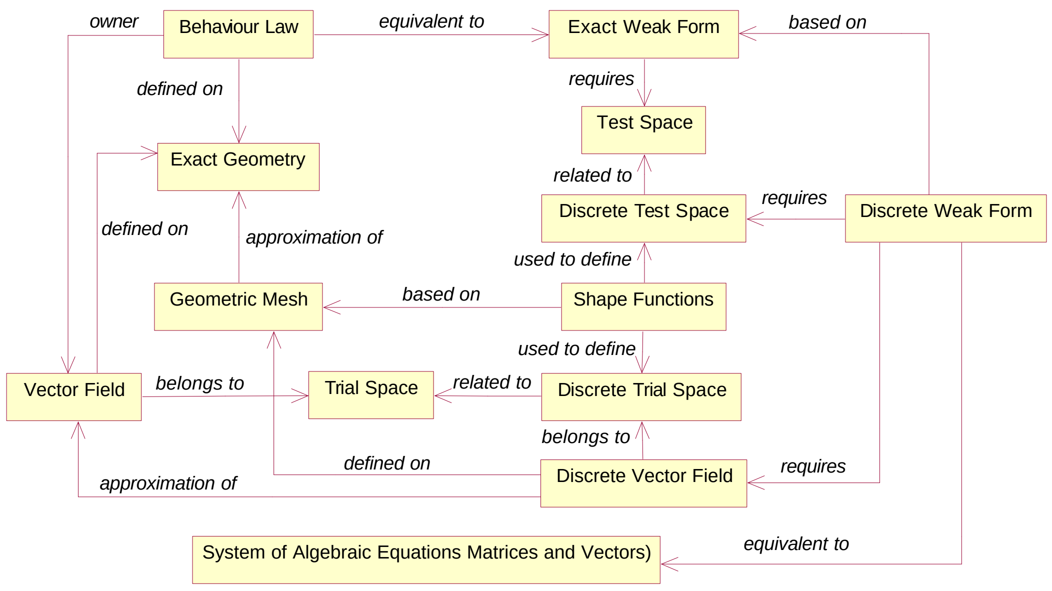


Figure 2-6 Summary of FEM concepts

In the next subsection we will detail the description of what composes the approximate (discretized) problem. Also we will consider the algorithm for the solution of the system of algebraic equations, which is obtained by the discretization process.

2.2.1 The Approximate Problem

By the *discrete formulation* of a phenomenon we mean the system of algebraic equations (either linear or non-linear) obtained through the application of a finite element discretization technique (discretization process) to the exact formulation.

A discretization process involves the approximation of the problem geometry (mesh of finite elements), the approximation of the phenomenon vector field and the approximation of the respective behaviour laws (see Figure 2-7). The most used method for storing the required information of the system of algebraic equations is using vectors and matrices. If the problem is time-dependent, a time progression

[1] This figure represents a UML class diagram which objective is only to represent the association between the major FEM concepts.

numerical scheme should be applied and the result is again a system of algebraic equations. If an algebraic system is linear, the vector-matrix symbolism is **K.d = r**, where **d** is a vector of unknowns (coefficients of the trial shape functions in the discrete vector field), **r** is a known vector, and **K** is a known matrix.

Typical FEM process:
1. Approximate a geometry with a collection of many geometric pieces of simple form (mesh generation);
2. Compute the phenomenon contribution (matrices and vectors) on each piece (finite element) to the discrete behaviour laws (system of algebraic equations);
3. Assemble those contributions from all elements together into an algebraic system of equations;
4. Solve that system.

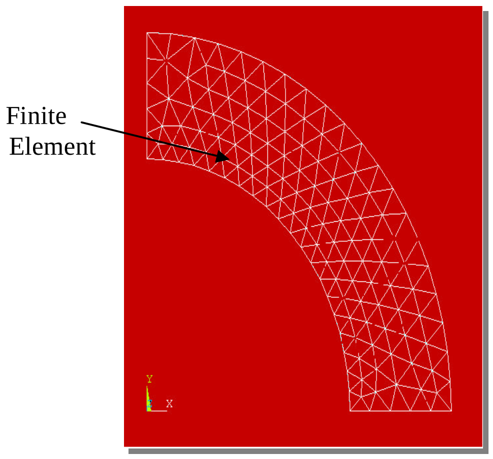


Figure 2-7 Typical FEM process

Even when a system of algebraic equations is non-linear, its solution frequently uses a solution of linear systems of algebraic equations, e.g. when using solution algorithms related to the Newton-Raphson method [ZP00]. Therefore, the explanation about the processes involved in solution algorithms for the FEM can be restricted to the assembling and solution of linear systems of algebraic equations. For those types of systems, a typical FEM program performs the following tasks:

i) Generates a mesh for the given geometry;

ii) For each finite element from a given mesh:
  - Compute a specific matrix (element matrix);
  - Compute a specific vector (element vector);
  - Assemble the element matrix in a large given matrix (global matrix);
  - Assemble the element vector in a large given vector (global vector);

iii) Solve the linear system of algebraic equations with the matrix as the global matrix and the global vector as the right-hand side of the eqaution.

Below each involved concept is detailed. As an example, we will consider the one-dimensional example defined in Figure 2-5.

a) Discretized Geometry

The exact **geometry** is composed of geometric entities, which could be 0-D (point), 1-D (curve), 2-D (surface) or 3-D (volume), embedded in spaces of either equal or higher dimensions. A phenomenon can be defined in any of these geometric elements.

The exact geometry is approximated by the union of simple geometric parts (geometric finite elements), with disjoint interiors. The set of all those finite elements is called the geometric mesh. Those geometric finite elements can be represented by edges, triangles, quadrilaterals, and tetrahedrons or hexahedrons, etc. In the example presented in Figure 2-8 the triangular geometric element is used to build the mesh. These elements can have distinct sizes and orientations, adapting themselves to the features of the original geometric domain. The so-called mesh generator automatically generates a mesh. There are several methods available for building a geometric mesh given a geometric domain. In order to look for the best numerical methods among the many techniques used to implement the FEM, it is very important for a simulation environment to be able to shift from one mesh generation method to another.

The geometry of an element is mapped into a known Geometric Reference Finite Element (**GRFE**), which is the same for a large set of those elements (usually, for all of them). The GRFE is defined with respect to a certain local coordinate system, and is responsible for providing data to important calculations. See Figure 2-8

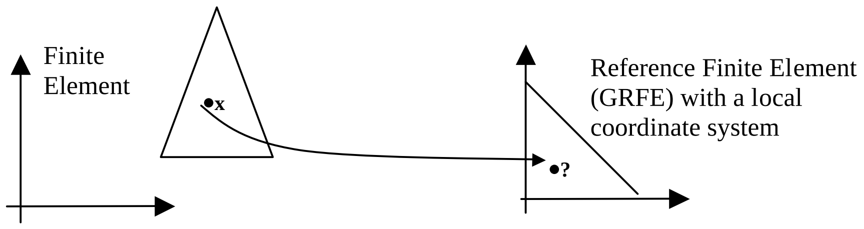

Finite Element
x
Reference Finite Element (GRFE) with a local coordinate system
?

Figure 2-8 Reference Finite Element Geometry

One example of the important calculations on the GREF is the numerical integration process, which is used to compute element matrices and vectors. The integration is originally defined over each finite element. However, it is transformed to the local coordinate system of the GRFE. Therefore, since the GRFE is fixed, the integration points and weights are independent from the finite element where the process was originally defined. Other pieces of data, important for numerical integration, are the inverse of the Jacobean matrix and the Jacobean, which depends on the mapping that map the finite element onto the GRFE. Those functions should be evaluated at the integration points. The GRFE is also important because it allows for the definition of

shape functions independently of a specific finite element. Therefore, the only pieces of data, which actually depend on a specific finite element are those related to the Jacobean matrix, as already mentioned. This approach simplifies tremendously the computation of matrices and vectors for each finite element.
In our one-dimensional problem, the considered geometry is a curve. The geometric finite element is represented by segment of a curve, that is, just an interval inside the geometric domain defined by the interval (0,1). For instance, we consider a mesh composed of three finite elements, as shown in Figure 2-9.

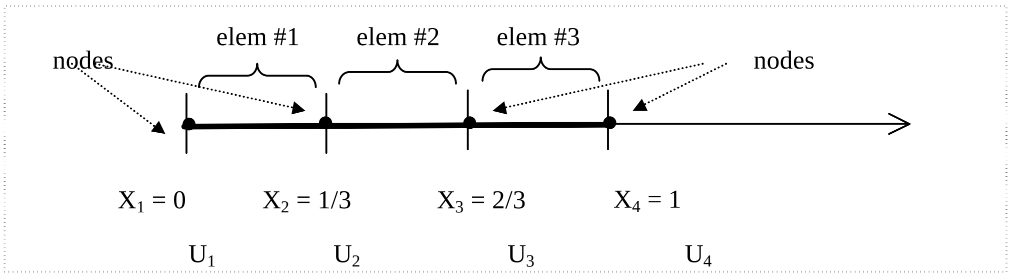


Figure 2-9 Mesh example used in the algebraic system generation

b) Approximate Vector Field

The discrete vector field means a vector field, which is a linear combination of the components of the known basis for the discrete trial space. Its coefficients are determined requiring that it should satisfy a certain discretized version of the exact behaviour law. The approximation of the phenomenon vector field is obtained using trial shape functions defined over a geometric mesh (see Figure 2-10 and Figure 2-11). They are built in such a way that it is possible to define a polynomial vector space of a specific order over each geometric finite element. High order shape functions can be used whenever a higher accuracy is desired.

$$u_h(x) = \sum_{i=1}^{n} u_i y_i(x)$$

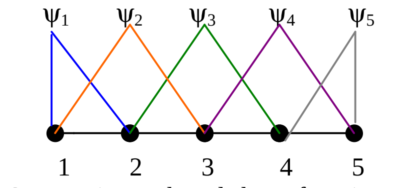
Figure 2-10 Example of Vector Field

The **mesh** definition is very important for the definition of the spaces of shape functions used in the approximation of the exact solution. When an order of approximation is given for a finite element of a given shape, the definition of the shape functions should be immediate. Relating the shape functions to vertices, edges, faces and volumes of the reference finite element does this. For instance, there will be shape functions related to the vertices, to the edges (which should vanish on the vertices), to the faces (which should vanish on the edges) and to the volumes (which should vanish on the faces). It is possible to prove that in this way one can obtain a basis for the polynomial space of the required order.

Example of linear shape functions:

$$y_1(x) = \frac{x_{i+1} - x}{x_{i+1} - x_i} \qquad y_2(x) = \frac{x - x_i}{x_{i+1} - x_i}$$

for x in between of x_i and x_{i+1}



Geometric Mesh and shape functions

Associate one and only one node to each vertex of the elements. Associate to each node one and only one shape function. Each shape function ? i, of node i, satisfies the following properties:

1. $y_i(x_j) = 0$ if i ? j and $y_i(x_j) = 1$ if i = j;

2. $y_i(x)$ is different from zero only on the finite elements linked to node i;

3. In the present case y_i is continuous and coincides with an affined function inside each finite element.

4. The sum of all shape functions is unity, that is the variable u_h (in Figure 2-10),can represent a constant solution within the element: $\sum_{i=1}^{2} y_i(x) = 1$

Figure 2-11 Shape Functions

As can be seen, there is a set of pieces of information, which is related to each shape function: the coefficient, which is called the nodal value and its dimension; the index of the shape function (indicating its order inside its associated geometric entity, i.e., vertex, edge, face or volume) and its connectivity (a number, which is important for the assembling of element matrices and vectors). All those pieces of information we relate to the notion of **node**. Thus for each shape function there will be a node carrying the correspondent data inside. The dimension of the nodal value is called the number of nodal degrees of freedom[2]. The sum of all the nodal degrees of freedom corresponds to the system dimension.

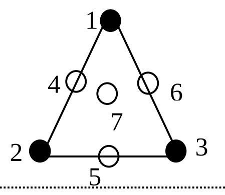


Figure 2-12 Abstraction of a Geometric Element and its Nodes

[2] By degree of freedom we mean the largenesses that will be user

Figure 2-12 shows an example of nodes represented in a triangle finite element (7 nodes are identified). Therefore, the polynomial space defined in this triangle is of order 2, plus an extra third order shape function, which is the seventh one.

Now, provided the shape functions are continuous over the whole domain, finite elements should share nodes whenever they share vertices, edges and faces, respectively. As can be seen, there is a one-by-one correspondence between the nodes and the shape functions. All the nodes of a mesh are numbered (connectivity) without missing any intermediate number. In this way the connectivity number defines the position in global matrices and vectors, related to the coefficient of the associated shape function.

In our example, the discrete trial and discrete test spaces will have dimension 4, because there are four nodes and the dimension of the nodal value is 1, see Figure 2-9. The approximation u_h is shown in Figure 2-13. However, if the boundary conditions are taken into consideration, we can conclude that $a_1 = 0$ and $a_4 = 0$. Therefore, the discrete trial and discrete test spaces can be restricted to a subspace of dimension two, spanned by $?_2$ and $?_3$. As a consequence, the discrete vector field culminates as shown in Figure 2-14. Those spaces can be restricted assuming the boundary condition described before in Figure 2-5.

Figure 2-13 Approximate Vector Field

$u_h = a_1 y_1(x) + a_2 y_2(x) + a_3 y_3(x) + a_4 y_4(x)$, where a_1, a_2, a_3 and a_4 are unknown constants to be determined

$u_h = a_2 y_2(x) + a_3 y_3(x)$, where a_2 and a_3 are unknown constants to be determined

Figure 2-14 Approximate Vector Field

The two functions, which span the two discrete spaces, are shown in Figure 2-15, which are equivalent to the ones presented in Figure 2-11.

$$y_2(x) = \begin{cases} 3x & 0 \le x \le \frac{1}{3} \\ \text{2-3x}, & \frac{1}{3} \le x \le \frac{2}{3} \\ 0, & \frac{2}{3} \le x \le 1 \end{cases} \qquad y_3(x) = \begin{cases} 0 & 0 \le x \le \frac{1}{3} \\ \text{3x-1}, & \frac{1}{3} \le x \le \frac{2}{3} \\ 3 - 3x, & \frac{2}{3} \le x \le 1 \end{cases}$$

Figure 2-15 Shape functions for the example

Phenomenon mesh is a set of data distributed over a given geometric mesh, which describes the approximation order of the discrete vector field on that mesh. It is comprised of a set of phenomenon finite elements – on a one-to-one relationship with the geometric finite elements - and related phenomenon nodes. Those nodes are related to the trial set of functions - which also have a corresponding meaning for the test space - of the phenomenon being considered. Each phenomenon finite element (of a phenomenon mesh) represents the order of approximation of the discrete vector field on that element.

In the example, the phenomenon mesh only indicates the polynomial order equal to 1 in each finite element.

c) Approximate Behaviour Laws

As an example, we consider the problems described in Figure 2-5. An equivalent formulation for the phenomenon behaviour laws is provided by the exact weak form, which are integral (see Figure 2-16) forms obtained from the (possibly Integra-algebraic) differential equations (see Figure 2-5). In the approximation method for this behaviour law, the first step is to select discrete trial and test spaces and assume an approximate solution *$u_h(x)$* (discrete vector field) in the discrete trial space, which depends on unknown coefficients to be determined and will substitute the exact solution *$u(x)$*. Once the approximate solution is defined, we substitute it into the weak form, which should be satisfied for all functions *$w_h(x)$* replacing *$w(x)$* in the discrete test space (see Figure 2-16). The accuracy of an approximate solution is dependent upon the proper selection of the discrete trial and test spaces. A discrete weak form represents the discrete behaviour laws for the discrete vector field.

$$I = \sum_{i=1}^{n} \int_{xi}^{x_{i+1}} \left(\frac{dw}{dx}\frac{du}{dx} + wu - xw \right) dx - \left[u'w \right]_{0}^{1} = 0$$

Figure 2-16 Exact Weak Form

Since the approximate solution depends on a finite number of unknown nodal values (coefficients) and since the test space is a finite dimensional, the result is a system of algebraic equations, which is solved for the nodal values. Note that the last term on the left side of Figure 2-16 represents values at the boundary of the geometric domain. Those values are defined by the boundary conditions, which should be prescribed for each simulation using the same behaviour law.

If what is to be solved is just a linear system – which happens to be the case - and the solution method does not require different auxiliary systems to be solved, the simulator may just assemble the matrix and the right-hand side of the system and apply a given method to solve it. However, it is very important to note that the solution method is the piece of information that defines which matrices and vectors are to be assembled, what system is to be solved and what information is to be used to carry out the processes. Thus, it is the solution algorithm, which will define what contributions a weak form should provide at each finite element.

Let us explain the above process with more details. Figure 2-16 shows the exact weak form, where *u* and *w* represent the exact solution and a generic test function, respectively. In this case, the discrete weak form is obtained by the substitution of the trial and test spaces by the discrete trial and test spaces, respectively, keeping the whole structure of the exact weak form. So, a member of the discrete trial space, the discrete vector field, substitutes the exact solution u_h. A generic member of the discrete test space, w_h, substitutes the test function w.

As we have seen in our example, the basis of the discrete test space has two functions *?*$_2$ and *?*$_3$. Since the dependence of the weak form with respect to the test function is a linear one, the discrete weak form can be defined by using only the components of the basis of the discrete test space. The consequence is that the discrete weak form can be represented by two equations, as shown in Figure 2-17, obtained by the substitution of functions *?*$_2$ and *?*$_3$, respectively.

$$I_1 = \int_0^1 \left(-\frac{dy_2}{dx}\frac{du_h}{dx} - y_2 u_h + xy_2 \right) dx = 0 \qquad I_2 = \int_0^1 \left(-\frac{dy_3}{dx}\frac{du_h}{dx} - y_3 u_h + xy_3 \right) dx = 0$$

$\left[y_i \frac{d\tilde{u}_h}{dx} \right]_0^1$ is omitted **i = 2, 3**, because ? $_2$(0) = ? $_2$(1) = ? $_3$(0) = ? $_3$(1) = 0, due to the prescribed boundary conditions.

Figure 2-17 Discrete Weak Form for each element

Figure 2-18 shows the transformation of the discrete weak form into an algebraic system of equations, by substituting the expression of u_h into the relations of Figure 2-17. Figure 2-18 shows the integration decomposed into a sum of integrals of each finite element. Thus, it illustrates the fact that one can build the same system of algebraic equations adding up contributions that come from each one of the finite elements.

$$I_1 = \int_0^{\frac{1}{3}} [-3(3a_2) - 3x(3a_2x) + x(3x)] + dx + \int_{\frac{1}{3}}^{\frac{2}{3}} [-3(3a_3 + 3a_3) - (2-3x)(2a_2 - 3a_2x + 3a_3x - a_3) + x(2-3x)]dx + \int_{\frac{2}{3}}^{1} 0dx$$

= -6.222 a_2 + 2.9444 a_3 + 0.1111=0

$$I_2 = \int_{\frac{2}{3}}^{1} 0dx + \int_{\frac{1}{3}}^{\frac{2}{3}} [-3(3a_2 + 3a_3) - (3x-1)(2a_2 - 3a_2x + 3a_3x - a_3) + x(3x-1)]dx + \int_{\frac{2}{3}}^{1} [3(-3a_3) - (3-3x)(3a_3 - 3a_3x) + x(3-3x)]dx$$

= - 2.9444 a_2 -6.222 a_3 + 0.2222 = 0

Solutions for a_2 and a_3 are a_2 = 0.0488 and a_3 = 0.0569. This is the approximate solution:

$$u = 0.0488 y_2(x) + 0.0569 y_3(x)$$

$$\begin{bmatrix} 6.222 & 2.9444 \\ 2.9444 & 6.222 \end{bmatrix} \begin{bmatrix} a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} 0.1111 \\ 0.2222 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Figure 2-18 Transformation to the Algebraic Linear System of equations

Whenever the solution algorithm asks for the solution of a system of algebraic equations, a loop is made over all the finite elements from its phenomenon mesh. The required vector and matrix quantities are calculated for each finite element. Those element wise matrices and vectors are assembled into the global matrix and vector. Afterwards the solution of the system can be solved for the nodal values (coefficients).

2.2.2 Algorithm for the Solution of Algebraic Systems

There is no single solution algorithm for solving the whole system. The one to be used should be formulated in detail and may require the decomposition of the whole solution in iterations that involve solutions of auxiliary problems (systems of algebraic equations), which will be effectively solved. The whole process may be decomposed into typical sub-processes:

- Production of a finite number of vectors and matrices, which are designated to be produced for each finite element and assembled into global matrices and vectors at certain stages of the solution algorithm. Such matrices and vectors may depend on discrete vector fields from other phenomena and, then is said to be coupled.
- Assembling of linear algebraic linear systems, which means the assembling of the matrices and vectors calculated for each finite element. The computations follow the definitions coming from the discrete weak form and use the shape functions, reference elements and integration rules needed to perform the tasks.

- Operations with vectors and matrices, that can occur at finite element level or not: linear combination of vectors and matrices, matrix-vector multiplication; vectors or matrices scalar product; vector vectorial product; multiplication of vectors and matrices by a scalar, etc. Many of these operations may be needed during the execution of processes defined by the solution algorithm,
- Solution of algebraic linear systems: the solution algorithm defines which type of solver is required for each linear system to be solved.
- After a final solution is obtained, posterior error estimation can be computed, which can be used for adaptation procedures: adaptation of the space and time discretizations, adaptation of the distribution of approximation order and adaptation of the distribution of models.

2.2.3 FEM Users

The typical FEM users ask what kinds of elements should be used, and how many of them. They can also ask: Can the model be simplified? How much physical detail must be represented? Is the important behaviour static, dynamic, non-linear or what? How accurate will the answers be and how will they be checked? The user must understand how elements behave in order to choose suitable types, sizes and shapes of elements, and to guard against misinterpretations and unrealistic high expectations. A user must also realize that the FEM is a way of implementing a mathematical theory of physical behaviour. Accordingly, assumptions and limitations of theory must not be violated by what the software supports. In some dynamic and non-linear analyses, algorithms by which theory is implemented must be carefully chosen, to avoid inappropriate algorithms, and to avoid interpreting results produced by algorithmic quirks or limitations such as actual physical behaviour.

A responsible user must understand the physical nature of the problem and the behaviour of finite elements well enough to prepare a suitable model and evaluate the quality of the results. Competence in using FEM for stress analysis does not imply competence in using FEM for (say) magnetic field problems. Engineers who use the software are responsible for results produced, not the software developer, even if results are affected by errors in the software.

An important problem, that users of numerical analysis techniques have to deal with, is computed discrete data set itself does not promote an immediate understanding of the behaviour of the quantities of interest, which were computed during the simulation. Commonly, analysis executed in personal computers can generate hundreds of megabytes of floating-point numbers. So, visualization techniques, when built in a suitable way, represent the key technology needed to extract information from large volumes of discrete data. There is a need to provide the interaction between the simulator and those pieces of information and visualization tools.

2.2.4 Typical Errors in FEM Solution

The three main sources of errors in a typical FEM solution are: (i) **Discretization error** results from transforming the physical system (continuum) into a FEM, which can be related to modelling the boundary shape, the boundary conditions, etc; (ii) **Formulation error** results from the use of vector fields that do not precisely describe the behaviour of the physical problem. For example, a particular vector field might be formulated on the assumption that it varies in a linear manner over the domain. Such an assumption will produce no formulation error when it is used to simulate a linearly varying physical problem, but would create a significant formulation error if used to represent a quadratic or cubic varying vector field; (iii) A **Numerical error** occurs as a result of numerical calculation procedures, and includes truncation errors and round-off errors. These errors occur at developer or user levels (for example, by specifying a physical quantity to an inadequate number of decimal places).

Important and relevant applications can be considered, when one is capable of adequately solving coupled phenomena systems, that is, a set of interacting phenomena, in space and time. The next section introduces and details these systems.

2.3 Coupled Phenomena

During the definition of a computational model for mathematical formalism, using the FEM in the context of coupled phenomena, the designer has to deal with problems such as data dependency and sharing. These issues are not trivial as to treat in a homogeneous way because they are very dependent on the specific problem being considered. It is difficult to provide reasonable high levels of abstractions, which could represent the main components, properties, relationships and operations involved. Without those abstractions, even when making use of sophisticated FEM libraries, the tasks involved in building and accessing new methods could become very costly and time consuming due to the lack of modularity and reuse. On the other hand, as far as we are concerned, there is no standardized solution for multi-physic systems (see chapter 1), that is, the control of coupled phenomena, making the integration of reusable code a very difficult task.

Examples of coupled phenomena, represented by coupled partial differential equations are: the displacement of a material body and temperature distribution in the same body. See equations 1 and 2 in Figure 2-19.

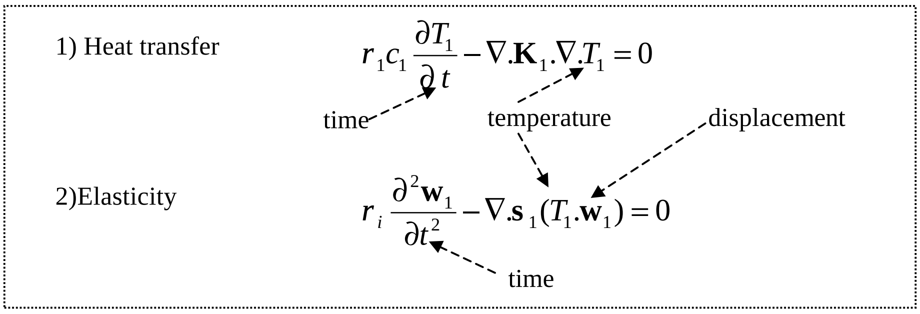

1) Heat transfer
$r_1 c_1 \frac{\partial T_1}{\partial t} - \nabla.\mathbf{K}_1.\nabla.T_1 = 0$
time
temperature
displacement
2)Elasticity
$r_i \frac{\partial^2 \mathbf{w}_1}{\partial t^2} - \nabla.\mathbf{s}_1(T_1.\mathbf{w}_1) = 0$
time

Figure 2-19 Examples of Coupled Differential Equations

The simulation of coupled phenomena involves many processes, which are described in the solution algorithm. A solution algorithm is tailored for the solution of the system of algebraic equations that is defined by the discrete behaviour law (in discrete weak form). A popular way of solving problems with many phenomena is dividing it into many problems involving only few phenomena (auxiliary problems). Thus, a sequence of procedures, which dictate the order in which each one of the auxiliary problems should be assembled and solved, is defined in the solution algorithm. Usually, this is an iterative procedure. Since the phenomena are coupled, the computations of matrices and vectors by a given phenomenon at the level of the finite elements may need evaluations of vector fields (at a given state) from other phenomena at the integration points (data dependency). Although different phenomena may share a geometric mesh (data sharing), this is not a requirement, making information management even more complicated. Those relationships are illustrated in Figure 2-20

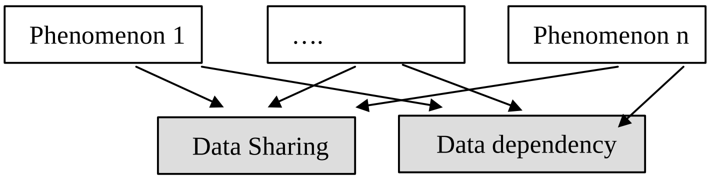


Figure 2-20 Phenomena Relationship

The FEM conventional process can be extended to treat coupled phenomena. One extension is the incorporation of information on vector fields from other phenomena, see Figure 2-20, and in the description of behaviour laws (weak forms) of a given phenomenon. Another important extension is the consideration of the solution algorithm as an input (data) to the problem. Thus, the processes described therein can be put into higher levels of abstraction through a hierarchical modularisation and pattern identification, improving reusability. There are other extensions, which due to simplicity are not detailed here.

An example of coupled multi-physics phenomena is water flow in estuaries and the chemical and biological (organic/inorganic chemical substances and living species) dynamics of the water quality. Some couplings are related to the evolution of chemical substances and biological species, which are coupled with the water flow and several predefined inputs to the system. Some more specific and detailed examples can be seen in Appendix A.

The solution of coupled phenomena problems can be used in different kinds of applications, which require modelling of different classes of simulators. A class of simulators can be defined by a set of pre-defined functionalities defined by a class of solution algorithms. For example, a class of solution algorithms may define processes specific for time-dependent phenomena possibly with model and discretization adaptivity. The class of problems a simulator can tackle can be defined by the types of discrete weak forms (behaviour laws) for respective types of discrete vector fields (describing the physical phenomena), which are defined on types of physical domains (geometry) and for which there exists known types of boundary conditions and initial conditions. The objective of the simulation is to compute some quantities of interest, which are functions of those discrete vector fields. The post-processor computes those quantities. It is important to mention that very frequently a post-processor procedure resembles tasks, which are typical of a phenomenon.

Running a simulation for a problem involving coupled phenomena consists of simulating a set of physical phenomena. The phenomena can be transient, i.e. time dependent, or otherwise stationary. Finite element meshes can be adapted during the simulation with the objective of keeping estimated errors below given tolerances. The simulation result – as already mentioned - is composed of discrete vector fields and post-processed quantities, which are also produced by special Computational Phenomenon objects. An important simulation (pre-processed) data, for each phenomenon, is its phenomenon and geometric meshes representing the spatial distribution of the approximation order and discrete geometry, respectively.

In the following sections, the main problems related to the development of simulators based on coupled multi-physic phenomena are identified and commented on.

2.4 Issues in the Development of FEM Simulators

The main issues related to the development of FEM simulators for coupled multi-physics phenomena are, on the one hand, the need for simulators that could simulate a large amount of coupled phenomena based on the FEM and, on the other hand, the need for computational environments or techniques, which could help the construction and configuration of those simulators.

However, other issues are also relevant:

- Difficulty to replicate published numerical studies. Experimental physics imposes on the other disciplines a formal way to describe an experiment, wherein basically all the information required to replicate it is defined and should be given. The replication of a given experiment, by multiple independent researchers, is a fundamental step in the establishment of a scientific truth. Purely theoretical studies allow any reader to access the experiment. To replicate it just one tool is needed, the researchers brain [VM02]. Among experimental studies there are the numerical studies. A numerical model relies on a variety of specificities associated to the numerical implementation of the theoretical model being used. Let's use, for example, the FEM model, which is the focus of this work. A FEM model relies on a theoretical model, which can be described exhaustively in writing, but it is very difficult to describe all the conditions involved in the universe of possible numerical methods. It is up to the designer to select the conditions and numerical methods that make sense and build the appropriate solution. Seldom, all those details, needed to replicate a numerical experiment, are published together with the experiment description.
- The question of reliability of computer-generated predictions is of great interest to specialists. Without some confidence in the accuracy of simulations, their value is obviously diminished. Today, remarkably accurate and reliable simulations are obtained routinely in many application areas while others are, at best, qualitative and capable of depicting only trends [COM00].
- The reuse of models, replication and extension. The Committee on Theoretical and Applied Mechanics emphasises that the selection of the mathematical and computational model is quite often the single most important step in obtaining valid computer simulation of physical events [COM00]. Model selection is a largely heuristic process, based on the judgment and experience of the modeller and on testing and experimentation. But it is frequently purely a subjective endeavour: different analysts may select different models to describe the same physical phenomena. Until now, there are few advances for making this selection step easier. It is quite relevant to support model reuse, replication and extension and persistence of previous simulation data and results.
- The achievement of higher levels of sophistication in simulation design can be obtained through the use of cooperative systems, where several technicians define and implement solutions together following the same patterns, thus improving the quality of the whole solution. The need for this cooperation is based on the fact that, frequently, there are stakeholders that are specialists in modelling and simulation of different classes of phenomena.

- In the simulation of coupled phenomena applications, specifically, there is a need for software packages, which could tackle the problem of multi-physics simulation (as can be seen in section 2.3). Most of the existing tools in this area treat each phenomenon independently, not achieving the required results, because sometimes the solution simply cannot be given in a partitioned form. Furthermore, without other alternatives, different software components have to be used together, when simulating coupled phenomena, giving rise to problems of data transfer and integration.
- Scientists and engineers require demanding tools to fulfil their research requirements and needs. There is an increasing need for flexibility in the construction of different solution strategies, support for the implementation of more suitable numerical methods, and a request for superior quality in simulation software component design, implementation and analysis. The main issue behind this is the fact that, due to the increasing complexity of the models and numerical methods, the construction of simulation software has become a major part of the scientists and engineer's work.

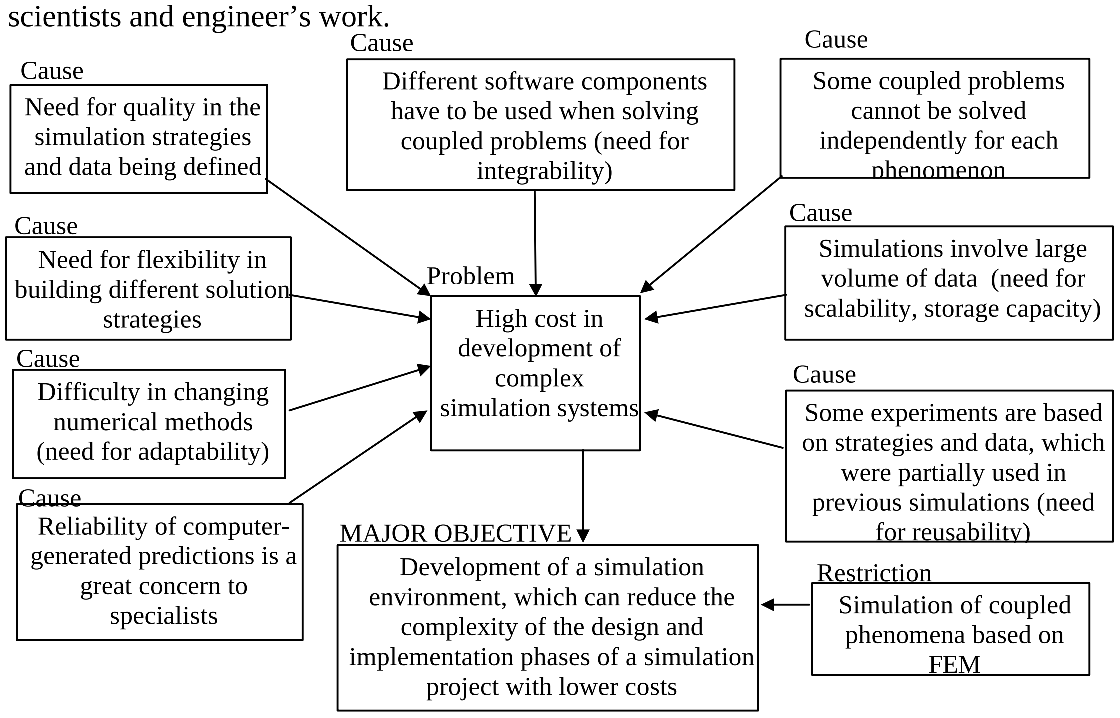

Cause
Need for quality in the simulation strategies and data being defined
Cause
Different software components have to be used when solving coupled problems (need for integrability)
Cause
Some coupled problems cannot be solved independently for each phenomenon
Cause
Need for flexibility in building different solution strategies
Problem
High cost in development of complex simulation systems
Cause
Simulations involve large volume of data (need for scalability, storage capacity)
Cause
Difficulty in changing numerical methods (need for adaptability)
Cause
Some experiments are based on strategies and data, which were partially used in previous simulations (need for reusability)
Cause
Reliability of computer-generated predictions is a great concern to specialists
MAJOR OBJECTIVE
Development of a simulation environment, which can reduce the complexity of the design and implementation phases of a simulation project with lower costs
Restriction
Simulation of coupled phenomena based on FEM

Figure 2-21 General Problem Identification – Existing reality

Some of the causes that justify the strong need for more powerful simulation development environments are presented in Figure 2-21. In order to improve the simulation software for coupled multi-physic phenomena, several objectives are required to be satisfied (see Figure 2-22).

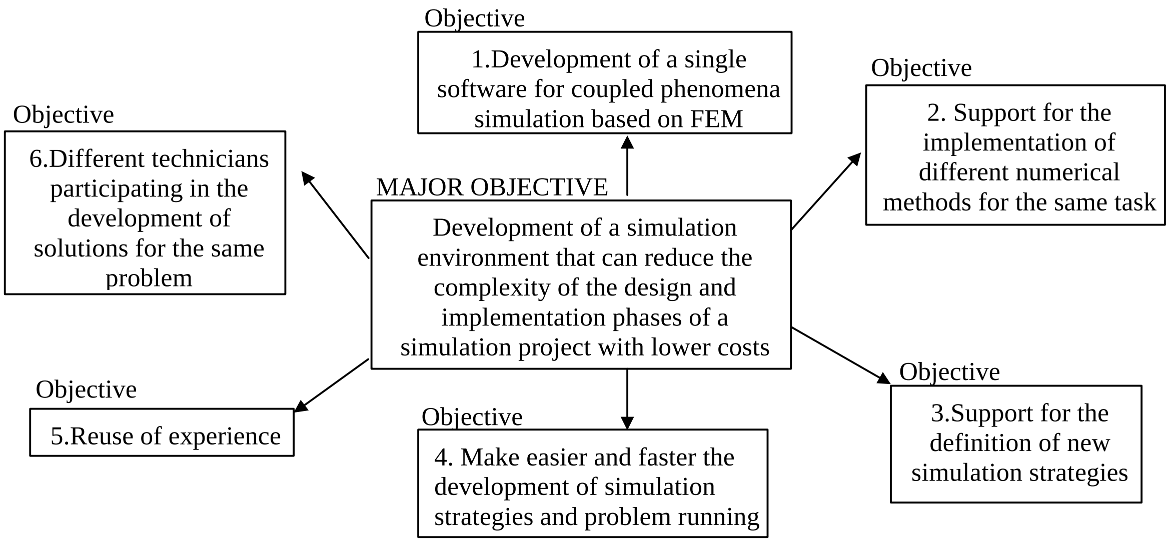

Objective
1.Development of a single software for coupled phenomena simulation based on FEM
Objective
2. Support for the implementation of different numerical methods for the same task
Objective
6.Different technicians participating in the development of solutions for the same problem
MAJOR OBJECTIVE
Development of a simulation environment that can reduce the complexity of the design and implementation phases of a simulation project with lower costs
Objective
3.Support for the definition of new simulation strategies
Objective
5.Reuse of experience
Objective
4. Make easier and faster the development of simulation strategies and problem running

Figure 2-22 Objectives towards an improvement in Simulation Environments

2. 5 Related Work

Due to the great relevance of simulation in different application areas, the community of simulation researchers is very active and we can name several existing initiatives, which are related somehow to our work (FEM simulation or other types of simulation). It is not the objective of this work to make a complete list of them, but we would like to refer to a few of them, due to their importance and achievements:

- DIFFPACK [LAN97] addresses an object-oriented strategy for the development of software for solving systems of partial differential equations (PDEs). The proposed strategy encourages reuse of modules capable of solving the involved sub-problems. It extends the basic ideas of an object oriented numerical library to a higher level where the objects reflect partial differential equations. It also opens the possibilities of building repositories of solvers for single PDEs that can be combined with each other in a flexible way. In [LAN01] a pre-processor is used to generate geometric input data required by FEMs. It also defines an abstraction for simulating coupled problems by operator splitting techniques.
- In [RM96] an algorithm framework for flexible FE based modelling is presented. Its goal is to explore the development of applications, accommodating the addition

of new modelling capabilities or the adaptation of existing ones, which can be applied by users. It offers families of algorithms that can be easily accessed and changed dynamically, providing algorithm flexibility. Distinct algorithms can be used in different parts of the model.

- [TO95] presents an overview of consecutive logical stages of technical aspects of the design process for modelling the behaviour the FEM structure. The process begins with the identification of the physical problems and design objectives and – usually after several modifications – yields a model of a product performing specific functions that satisfies design criteria. During the design process, knowledge about the model is enriched, modified, and used at further stages of the design. The simulator process development using FEM is composed of: Understanding a physical problem; Mathematical Modelling of the structure; Discretization of the Model; Selection of computational methods and strategies; Numerical analysis of the discrete model solution; Generalized pos-processing; Verification of the finite element solution; Modification of the model, that is, the construction of problem specification or variation in model optimisation; and accumulation of experience.
- SCIRUN, a scientific programming environment that allows the interactive construction, debugging and steering of large-scale scientific computations [PWC97]. SCIRun allows scientists to: (a) design and change simulations interactively using a dataflow programming model; (b) design and change and view the geometry model; (c) change interactively simulation parameters and boundary conditions; (e) change the level of mesh adaptation needed to make a more accurate numerical solution and (f) visualize interactively simulation results. O SCIRun gives access to already built modules, integrate scientific libraries and also allows the development of new modules. It uses dynamic sharable libraries to allow the user to recompile only a specific module without having to make a complete re-link. The application programmer has the responsibility to break the application into suitable components, and also has to guarantee that the parameters change make sense considering the underlying physical problems. Modules represent computational algorithms or operations, where a set of input and output ports defines its external parameters. A port is a connection point to data routed for different stages of connection. Ports can be added or removed of a module dynamically. Output ports can maintains a cache for sets of data, avoiding re-computation. Output ports can be connected to several input ports. However input ports accept only a single connection. The process of writing a new module involves writing a new C++ class.
- FEMLAB [FEL04] is an interactive environment for modelling and simulating scientific and engineering problems based on partial differential equations (PDEs)

– equations that are the fundamental basis for the laws of science. With FEMLAB's multi-physics features, you can simultaneously model any combination of physics by choosing from these modelling approaches: (i) use the ready-to-use application modes to create a model by directly defining the physical quantities rather than the equations; (ii) Use equation-based modelling to have the freedom to create custom equations;(iii) Combine both approaches for multi-physics modelling. FEMLAB offers CAD tools, interfaces for physics or equation definitions, automatic mesh generation, equation solving, visualization and post-processing – all in an integrated environment. With the package's MATLAB interface you can extend the FEMLAB models through a powerful technical programming environment. The combination of an easy-to-use graphical user interface and the flexible programming capabilities make FEMLAB an unprecedented package for multi-physics simulations.

In appendix B some other existing approaches for simulation software development are described.

Most of these works support good practices and fundamentals, useful for defining worthy simulation systems solutions and capabilities. However, despite the richness of their contribution, in the case of the FEM solutions, we did not find an integrated environment that satisfies the demanded features for coupled phenomena environments described in earlier sections. There are still important questions related to abstractions of numerical algorithms that remain unsolved, such as: (i) mechanisms to allow easy interchange between numerical methods and strategies; (ii) repository organization and management and its relationship with simulation instances; (iii) satisfactory abstractions of couplings, which could be defined independently of the actual implementation of the participating phenomena; (iv) abstractions of groups of phenomena, which are to be solved together, making it possible to organize operator splitting strategies in different forms; (v) higher abstractions of the phenomena-geometry relationship, in order to decide whether each phenomenon will or will not share its mesh with other phenomena (defined in the same geometry); etc.

2.6 Final Considerations

There are many advantages in the use of the FEM for the simulation of coupled multi-physics problems, as the method can handle: complex geometry; many phenomena in a homogeneous way (such as heat transfer, solid and fluid mechanics); complex analysis types (steady and transient states, linear and non-linear behaviour laws); complex geometry-related data: (boundary data, domain-embedded data); and complex constraints. In addition FEM can handle bodies comprised of non-homogeneous materials: material properties may be given as functions or the geometry may be divided with each component being assigned different material properties. Special material effects can be handled, for example temperature dependent properties, and plasticity, etc. Special geometric effects can be modelled (such as large displacements, large rotations, and contact conditions).

For the development of FEM simulators a powerful computer and reliable FEM software are essential. The input and output data may be large and tedious to prepare and interpret. The FEM pre-process is also susceptible to user-introduced modelling errors: (i) Poor choice of element types; (ii) Distorted elements; (iii) Geometry not adequately modelled.

The main issues in FEM simulation systems development are related to their complexity and the consequent requirements of high investments of time and money, which can frequently make their development impracticable. Even though there has been great advances in software engineering in the past years, in FEM context there are only a few simulation tools and techniques that support high levels of abstraction, software reuse, friendliness, and maintainability of simulators based on the FEM for coupled multi-physic phenomena simulation.

In our work, we analyse the existing requirements in this specific domain of knowledge. We consider current trends in software engineering development and propose an environment to support FEM simulator development for coupled multi-physic simulators. In this thesis we define a structure, which improves the quality of simulator designs. The approach tries to avoid problems that are intrinsically related to some symptoms of poor design [MRC02] when: it is hard to change; it is hard to reuse; it is hard to do the right thing; there is needless complexity; there is needless repetition and disorganized expression of the world. The result is the proposal of a deep analysis of the problem domain, and the definition of architecture based on some specific patterns, which try to fill the existing gap in the development of FEM simulators.
In this thesis, FEM specific solutions can be gathered in a Simulation Environment for the development of FEM simulators. Many applications are to be developed, within the specific domain; so the timesaving of reuse will recover the time invested to develop it. However, developing a reusable system requires abstractions that do not become obvious until the application examples are tested in many situations. This work results from the experience obtained during the implementation of several simulators in the FEM context, from which important abstractions have been defined. These implementations were evaluated and from this experience conclusions have been drawn, which are used in our analysis.

Chapter

3

Analysing and Describing FEM Simulators with Problem Frames

The development of FEM simulators is complex and the search for standard solutions for its development is appropriate. This chapter improves the description of the specific domain of FEM simulators through requirement analysis and the problem domain specification. The existing world without the existence of the Plexus solution is presented, and then the proposed environment with the respective functional and non-functional requirements is specified. In order to help developers with problem analysis, decomposition and description of FEM simulator solutions, some specific classes of problem solutions (problem frames) are proposed. These patterns include meta-models, which help to define the involved context.

This chapter also investigates the suitability of the Problem Frames technique, which is applied for the improvement of the description of our specific domain of FEM simulators. The study evaluates the appropriateness of the technique, discussing its power of expressiveness and limitations and suggests what can be improved.

3.1. Introduction

The development of low cost and high quality complex software is a continuous challenge for the software engineering community. Different techniques, methodologies and frameworks have been proposed to improve software quality. Independently of the nature of the software, the elicitation, analysis, negotiation, specification and management of requirements are fundamental for the development of quality software.

This work focuses on the conceptualisation and design of a simulation environment to support the development of simulators. In this context we must identify and use a flexible technique to describe the problem domain, that is, the real world (physics, mathematics, chemistry, biology, etc.) and the FEM concepts, as well as for specifying the requirements for the simulators to be developed. For this purpose we applied the Problem Frames [JAC01] technique. One of our objectives is to evaluate the use of this technique in the context of FEM simulators environment conceptualisation.

We can summarize Problem Frames as a software engineering technique appropriate to the description of problem domains, defining requirements and decomposing the problem into sub-problems. A problem domain is composed into several concepts (such as entities, events and states) that we can observe in the real world and also the relationships between them. Some domains are controllers and others are controlled, and others are simply symbolizations of concepts and their relationship. Collections of these concepts can be grouped forming a part of the world that can be distinguished and conveniently considered – a domain.

This chapter is organized in the following way. Section 3.2 presents Problem Frame technique. Then, using the described concepts, in section 3.3, we first locate and limit the scope of the problem (which motivates the development of a FEM simulator environment), clarifying the distinction between the machine to be developed and the world, and the relationship between them. In section 3.4 the proposed environment (machine) is described, exploring the decomposition of the involved problems into smaller and simpler sub-problems. In addition the involved domains[1] are detailed. Furthermore, in section 3.5 some of the machine sub-problems are described; they are specific to the area of FEM simulator development. This definition is made by the identification of common patterns in the simulators context and the recognition of elementary problem frames. Section 3.6 evaluates the application of Problem Frames to the FEM simulator domain. Finally, in section 3.7 remarks about this chapter are made.

[1] Despite the order of the chapter sections, the problem domain (independent of the machine) and the requirements can be explored iteratively.

3.2. Problem Frames Technique

The Problem Frames [JAC01] technique objective is to help in progressing from problem identification to problem structuring, while focusing on the *domain* concept. *Domain* is a particular part of the world that can be distinguished because it is conveniently thought of as a whole, and can be considered, to some extent, separately from other parts of the world. The technique emphasizes the separation of problem and solution domains.

A general belief is that one should focus on the problem before the solution, that is, one should focus on *what* the system will do, before focusing on *how* it will do. However, it is often hard to distinguish a problem from its solution, nor any easier to distinguish *what* from *how*. The Problem Frames technique considers that it is more helpful to distinguish *where,* that is, to recognize that the solution is located in the computer and its software, and the problem is outside in the world. The computer can provide solutions to these problems because they are connected to the world outside. The connections between the computer and the world enable the computer to play its role in the solution, see Figure 3-1.

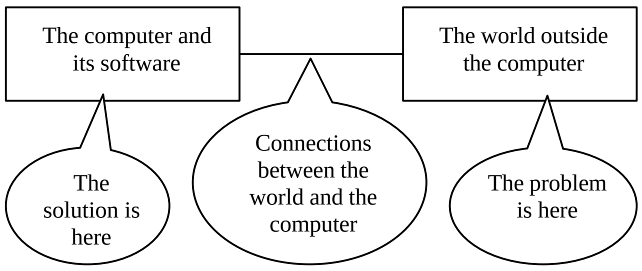

The computer and its software
The world outside the computer
The solution is here
Connections between the world and the computer
The problem is here

Figure 3-1 The Environment - Problem and the Solution

Sometimes the word "system" is used for the whole combination of the world and the computer together. So it is quite appropriate to say that the first step is to describe the system: in a wider sense, the system is where the problem is. But the word "system" also has the narrow sense of being the computer and its software. So there is an ambiguity between the wide and narrow sense. So, Jackson [JAC01] uses the term machine instead of system. A machine is that piece, which must be eventually built and installed to solve a problem.

Developing software is building a machine to solve a problem in a given domain to meet customer's needs, requirements. The machine is a general-purpose computer specialized by software. Problem decomposition gives rise to many sub-problems and also many machines [JAC95]. Note that the machine in one sub-problem may be a part of the problem domain in another sub-problem. The customer's requirement (property or behaviour) is in the problem domain. Requirements add constraints to the domain's intrinsic properties or behaviour. The machine is the solution and the problem is outside the machine. The machine and the problem interact at the interface defined by their shared phenomena. By phenomena we mean elements of what we can observe in the world, such as events and states.

The machine, the world and requirements are the main part of the software development problem. The solution task is to construct a machine so that the interactions with the world will ensure satisfaction of requirements. The existing problem in the world is described through indicative properties (what is known). On the other hand, the requirement and the machine are described as selected options, that is, what is being desired and planned to solve the existing problem in the world.

3.2.1. Indicative and Optative Moods

The environment is defined as the portion of the real world relevant to the software development project [JAC96, JAC01]. The requirements' specification includes statements in the:

- *Indicative mood*: this describes the environment as it is in the absence of the machine or regardless of the actions of the machine. This mood describes the domain of knowledge. It indicates the objective truth about domains (what is true regardless of the machine's behaviour). For example, the definition of what composes the formulation of a FEM simulation for multi-physic phenomena (as described in chapter 2).
- *Optative mood*: this describes the environment as we would like it to be and, as we hope it will be, when the machine is connected to the environment. It can be separated in *requirements* (options that customer has chosen) and *specification* (machine desired behaviour). For example, the system requirements for implementing configurable FEM simulators.

The indicative properties are at the heart of one's analysis. One relies on the domain properties to bridge the gap between specification of phenomena, that the machine can directly sense and cause, and the required phenomena that the customer is interested in. So, our subject includes problems, not only solutions. Since problems are in the world, being more precise about problems and their domains means being more precise about the world and its phenomena.

3.2.2. Phenomena

Jackson [JAC01] suggests that we must understand appropriate abstractions of phenomena, that is, of what we can observe in the world. This is necessary at specific levels of individual problems and domains. A phenomenon can be classified into three types: (i) Individuals, which are phenomena that can be named and distinguished from every other individual (a concept which has a specific meaning, structure and behaviour, so its features can distinguish it from other entities), examples are cars, or events identification such as the first time a person drives a car; (ii) Relationship, an association among two or more individuals (for example, a car revision represents a relationship between a broken car and a fixed car); (iii) any pattern or structure among phenomena of a domain.

Individuals can be further classified into:

- *Entity*, an individual that is mutable over time, that is, an individual that persists over time and can change its properties and states from one point in time to another. Examples include cars, people, and so on. In our work, examples are simulators and geometries.
- *Event*, an individual that is an occurrence at some point in time, regarded as atomic and instantaneous. For example, pressing a lift bottom, or in our work the process of starting a simulation.
- *Value*, an individual that is not subject to change, that is, an individual that exists outside time and space. For example, integers and strings. In our work, the values (strings) that determine physical phenomena context, such as "heat transfer" and "elasticity".

Relationships can also be further classified into:

- *States*, that is, time changing relations over non-event individuals (which can be true at one time and false at another, also an element of a state transition diagram), for example, the fixed car state. In our work an example is the simulator built state.
- *Truths*, unchanging relations over non-event individuals (a relationship that is either true at all times or false at all times). Then individuals are always values, and the truth expresses mathematical facts. One example is GreaterThan (5, 3).
- *Roles* represent the participation of individuals on events, in other words, a relation between an event and individuals that participate in it in a particular way. Each role expresses what one might otherwise think of as one of the arguments of the event. Roles are fixed; they do not change over time. In the event fix a car we have the roles of the object to be fixed (the car) and of the mechanic (the person who will fix it).

One can also distinguish two categories of phenomena:

- *Causal phenomena*, which include events, roles and states (relating entities). They are directly caused or controlled by some domain, and they can cause other phenomena in turn. Examples are input/output devices, arithmetic and logical units, buttons, lights, sensors, motors. In our work an example is the simulation, which is caused by a simulator, and in sequence causes simulation results;
- *Symbolic phenomena*, which include: values, truths, and states (relating only values). They are used to symbolise other phenomena and relationship among them. Some examples are input and output, a database held on one or more disk drives, an object structure inside a machine, a file held on a tape. In our work we have symbolic data representing stored data about existing kinds of physical phenomena or geometries.

Jackson gives a symbol for each kind of phenomena. For example, Y represents symbolic phenomena (for example, Y4); C represents causal phenomena (for example, C2).

Plexus phenomena are natural physical phenomena, so they are included as a subset of Jackson's phenomena. We must be careful here not to generate confusion. Note that Plexus refers to natural phenomena (a computational abstraction of physical phenomena, such as elasticity, heat transfer, rigid body motion, and so on).

3.2.3. Domains

A domain can be thought of as a collection of related phenomena, for example the simulator domain is composed of phenomena like simulator's strategies, the way natural phenomena are grouped and organized to be solved, simulators skeletons, and so on. Domains may share phenomena. Indeed the only way two domains can interact is through the interface of shared phenomena.

Based on phenomena categories, Jackson distinguishes different kinds of domain [JAC96, JAC01].

- *A Causal domain* is a domain whose properties include predictable causal relationships among causal phenomena (include events, roles and states); these relationships allow one to calculate the effect of the machine behaviour at an interface with the domain. An example is the car mechanics shop domain, which can start events such as car repair and painting and which includes states such as car repaired.
- *A Biddable domain* usually consists of people such as operators or users. In our work we have, for example, the simulator designer.
- *A Lexical domain* provides the significance of data, for example, input and output, a database etc. A definition can be found in [JAC96]: a physical representation of a symbolic phenomenon such as data, for example a database about simulations knowledge.

By considering the defined concepts of existing *problem domains* (which define the indicative properties) and the *requirements* and the *specifications* (the optative part of the desired machine), Jackson [JAC01] defines what he calls Problem Frames.

3.2.4. Problem Frames Diagrams

A problem frame defines an intuitively identifiable problem class in terms of its context and the characteristics of its problem domains, interfaces and requirements [JAC01]. A general representation of a problem frame is presented in Figure 3-2. The scripted rectangle represents the machine one wants to build. The plain rectangle represents the part of the world that interacts with the machine. The solid line connecting the two rectangles represents an interface of shared phenomena (for example, shared events and shared states). The dotted ellipse represents the requirement; the dotted arrow indicates that the requirement is a description, which is a predicate over the phenomena of the world.

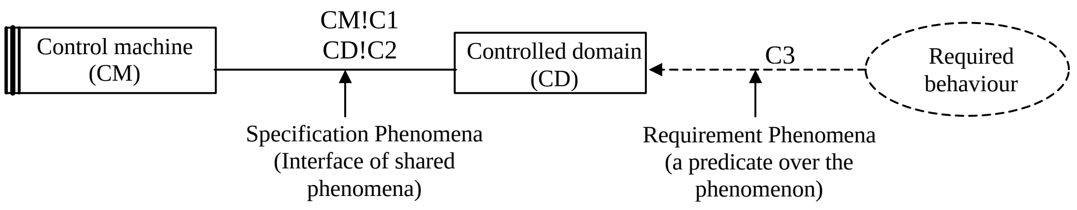

Control machine (CM)
CM!C1
CD!C2
Controlled domain (CD)
C3
Required behaviour
Specification Phenomena (Interface of shared phenomena)
Requirement Phenomena (a predicate over the phenomenon)

Figure 3-2 Generic Problem Frame Diagram

A frame diagram is just a slightly fancier generic problem diagram. It is different from an ordinary problem diagram in the following ways:

- The names of the parts are chosen to suggest their involvements in the general form of the problem: control machine, controlled domain and required behaviour (see Figure 3-2).
- The sets of interface and reference phenomena are denoted by short stylised names, like the names C1 and C2 (Figure 3-2). The names on interface connections also include the usual control prefixes (like CM for control machine and CD for controlled domain).

The control machine (CM) is the machine to be built (Figure 3-2**)**. The controlled domain (CD) is the part of the world to be controlled. The requirement, giving the condition to be satisfied by the behaviour of the controlled domain, is called the required behaviour.

In Figure 3-2, the interface of shared phenomena with the machine consists of: C1, which is controlled by the machine (CM), and C2, which is controlled by the controlled domain (CD). The machine affects the behaviour of the controlled domain through the phenomena C1; the phenomena C2 provides feedback. The requirement is expressed in terms of C3 phenomena of the controlled domain. These are the requirement phenomena. In general C3 will be different from C1 and C2. This gap must be bridged by indicative domain properties by the controlled domain.

A *concern* is an aspect of a problem demanding the developer's attention. For example, the completeness concern ensures that a description is complete and the initialisation concern ensures that the machine and the problem domains are in appropriate states at the start of the execution. Each frame has a *concern* that must be addressed in any problem of the class. The *concern* identifies the descriptions one must fit together properly in a correctness argument: requirement, specification and domain. In conjunction with the characteristics of problem domains, the frame concern gives rise to the particular concerns that distinguish the problem class. If one tries to fit a problem into an inappropriate class, the resulting development will certainly be awkward and probably unsuccessful.

Jackson also describes the concept of a *composite frame*, a familiar class of problems that demands decomposition into sub-problems in accordance with a standard structure. The sub-problems have frame concerns and other particular concerns; interaction amongst them gives rise to fresh composition concerns. The possible combinations of simple sub-problems are unlimited, but many composition concerns can be identified by examining some of the combinations in terms of problem domains common to different sub-problems. These concerns include *consistency* (between indicative or optative domain descriptions), *precedence* (between inconsistent domain descriptions), *interference* (between different interactions with a domain) and *synchronism*.

Problem Frames take part of the problem decomposition. Generally before its definition a context diagram is built for the computational system being defined, this diagram will influence the identifications of the system required problem frames.

3.2.5. Context Diagram

The Problem Frames *context diagram* is the first representation of the proposed problem [JAC95]. It is fundamental to determine where the problem is located, and what parts of the world it concerns. It gives an opportunity to structure the problem as a number of separable domains, together with the machine to be built, and to show how the domains interact with each other and with the machine. The structuring of the problem context is an essential step towards problem analysis, and choice of each sub-problem.

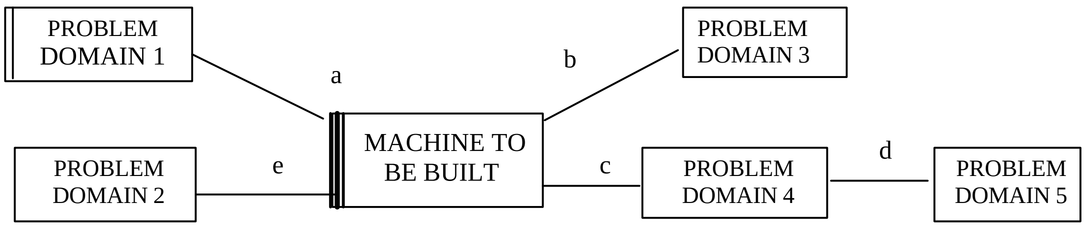


Figure 3-3 Context Diagram

Figure 3-3 exemplifies a generic context diagram composed of the machine to be built and five problem domains; it also shows the interfaces between domains, that is how the machine is connected to problem domains and how problem domains are connected to each other (through interfaced phenomena a, b, c, d and e). The domain with a single vertical stripe indicates that it is a domain that the developers must design (e.g. DOMAIN 1 in Figure 3-3). The other problem domains with no stripe are all given parts of the world. All the domains in the context diagram are physical. The interfaces can be understood as events, states and values shared between domains. By physical domain we mean parts of the world where the customer can check for observable effects. It can be divided into:

- *Machine domain*: this is the computer program which one must design and build;
- *Designed domain*: this is the physical representation of some information, for example stored on: a magnetic stripe card, or a tape or a floppy disk or a hard disk, or even on a screen or printed output. It is a description or model that the

developer is free to design (free to design and specify its data structure, to some extend, its data context;
- *Given domain*: this is a problem domain whose properties are given, that is, we are not free to design this domain. However, this does not mean that it already exists when you start the problem, only that you have information about its definition.

According to Jackson [JAC01], the following issues must be considered, although he does not impose order on them:
- Locating and bounding the problem, that is, expand and clarify the distinction between the machine and the world, and the relationship between them. The world is always structured as a collection of interconnected domains, pictured in a context diagram. Customer's responsibility and authority bound the problem context. The built machine must not change the parts of the world that the customer has not authorized, and in analysing the problem one must not ignore relevant parts of the world for which the customer is responsible.
- Explore the decomposition of problems into smaller and simpler sub-problems. Each decomposed sub-problem has its own projections – its partial views – of the world and of the machine, taken from the original problem. It shows each sub-problem in a *problem diagram*. That is like a context diagram with the addition of a requirement.
- After one has roughly identified what the problem is about, the next step must be to look into it more deeply. One can look more closely at the interface between the world and the system. For example, it could discover, decide, analyse or design more details of the messages in data-flows, or more detail of the dialogues for the use cases. Also the terminals and the actors can be explored, and there is no reason to restrict it to what can be seen at the interface with the computer.

Note that one does not have to draw the context diagram before analysing the requirements. It must explore the context and the requirements iteratively.

3.2.6. Repertory of base Problem Frames

Jackson identifies a repertory of five base problem frames [JAC01], which are recognized problem classes, with associated characteristics and solution methods. Within these structures, specializations can emerge and incremental advances be obtained, which otherwise could not be achieved by attempts on a more abstract or a broader front. The repertory of elementary problem frames, available in [JAC01], includes the following intuitive notion of each problem[2]:
- *Required Behaviour*: "There is some part of the physical world whose behaviour is to be controlled so that it satisfies certain conditions". The problem is to build a machine that will impose that control;

[2] There are some variants of these basic problem classes, most of them result from adding further domains to the problem world. All these variants raise additional characteristic concerns in the structure.

- *Commanded Behaviour*: “There is some part of the physical world whose behaviour is to be controlled in accordance with commands issued by another operator”. The problem is to build a machine that will accept the operator’s command and impose the control accordingly;
- *Information Display:* “There is a part of the physical world for which certain information (about its states and behaviour) is continuously required.” The problem is to build a machine that will obtain this information from the world and present it at the required place in the required form;
- *Work-pieces*, “A tool is needed to allow a user to create and edit a certain class of computer processable text or graphic objects, or similar structures, so that they can be subsequently copied, printed, analysed or used in other ways”. The problem is to build a machine that can act as this tool;
- *Transformation*, “There are some computer readable input files whose data must be transformed to give a certain output file. The output data must be in a particular format, and must be derived from the input data according to certain rules”. This machine will produce the required outputs from the inputs.

In the next section we apply Problem Frames for Plexus problem description – the Indicative description - also called knowledge domain. Then, section 3.4 presents the Plexus machine to be developed – the Optative description.

3.3. Locate and limit the scope of the Problem in the world (Indicative)

When engineers want to develop a new simulator/simulation based on FEM they must have a clear notion of the involved abstractions and their relationships, the concerns of the stakeholders, and reusability of the data and code. Generally they have to deal with the problem from the beginning, describing FEM abstractions, processes details and requirements for the new simulator. Furthermore, the multidisciplinary and interdisciplinary nature of multi-physics problems, induce unorganised domains, which adds to the great number of possible ways for knowledge representation, processing and use. High levels of abstraction and problem decomposition are required, in order to achieve a state where software reusability, maintainability and adaptability become commonplace together, with the possibility of easier software evolution. However, it is difficult to identify which kind of requirement modelling technique or architectural abstractions are relevant, appropriate and complete to understand the whole problem, especially if we aim at reuse and evolution of solutions.

An engineer using a FEM simulator can perform an analysis such as the one defined in Figure 3-4. Some requirements for performing a specific FEM-based simulation analysis are described by [COR95]:

i) A problem must be solved; problem data and simulation objectives are defined at the application level;

ii) Develop a plan for an initial Finite Element (FE) model, that is, define the problem data, discretization and solution methods;

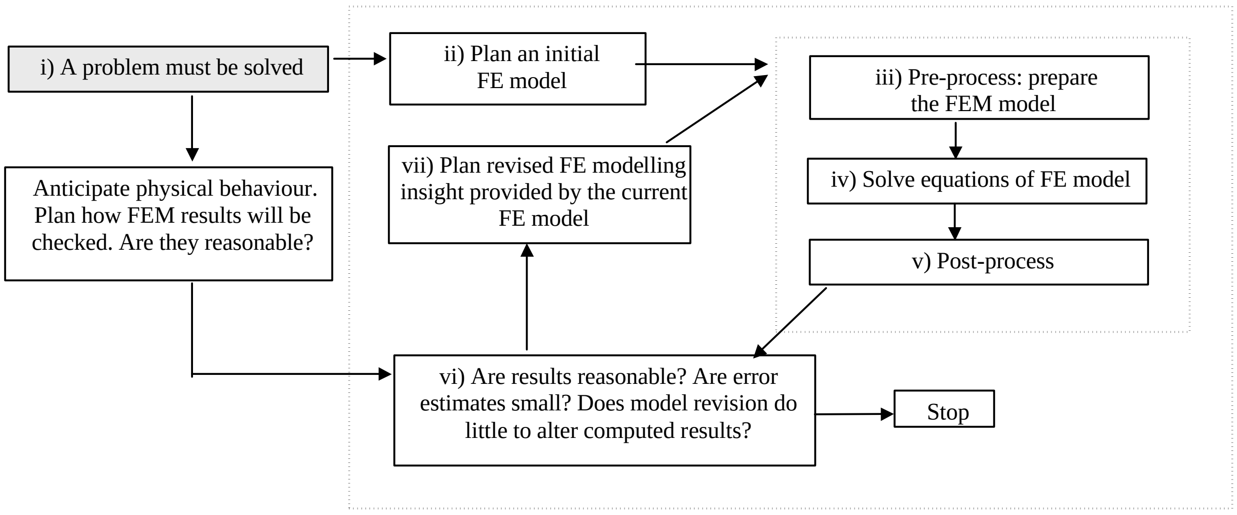
iii) Pre-process, which means processing the problem's input data, to obtain a set of data structures, which are needed for the simulation process;

iv) Execute the solution algorithm, which means a sequence of operations that represent the simulation itself;

v) Post-process, which means further processing the FEM simulation results in order to compute (based on the simulation results), store and display the required data.

vi) Validate the results. The physical behaviour must have been anticipated.

vii) Revise the plan if needed.



i) A problem must be solved
ii) Plan an initial FE model
iii) Pre-process: prepare the FEM model
Anticipate physical behaviour. Plan how FEM results will be checked. Are they reasonable?
vii) Plan revised FE modelling insight provided by the current FE model
iv) Solve equations of FE model
v) Post-process
vi) Are results reasonable? Are error estimates small? Does model revision do little to alter computed results?
Stop

Figure 3-4 Conventional FEM Analysis Process [COR95]

FEM design and procedure can be clearly divided into two types of processes: one which involves performing large-scale algorithmic computations and data processing; secondly processes involving decision-making, which require perception, intelligence, knowledge and reasoning power [TO95]. Engineers, expected to master the expertise necessary to use finite element software effectively in the design process, traditionally perform the decision-making process. The human expertise in the decision–making process represents a major part of the time and effort to perform analysis and design. To overcome these problems, several research efforts are currently underway, attempting to formalize the decision-making criteria and to develop intelligent automated software to supplement the human designer [TO95]. The problem we want to solve is related not only to decision making-process but also to ways of organizing processes and data (related to large-scale algorithmic computations and data processing) in a way that will facilitate further exploration of process and data distribution and reuse.

3.3.1 Plexus Indicative Mood

The world under study for the development of a new application, without the Plexus System (machine), can be structured as a collection of interconnected domains. In Figure 3-5 the problem domain, related to the development of simulators using the FEM, is organized as a number of separable domains, commonly found in most conventional FEM simulator worlds (based on Figure 3-4 which documents the domains of study).

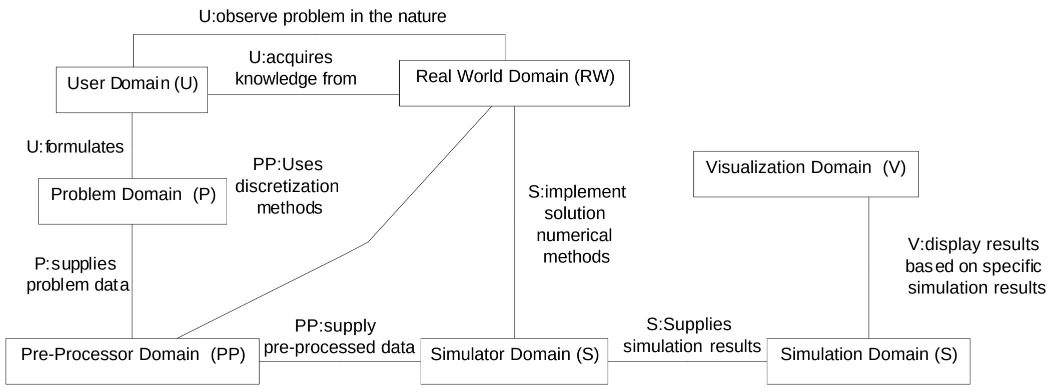

U:observe problem in the nature
User Domain (U)
U:acquires knowledge from
Real World Domain (RW)
U:formulates
PP:Uses discretization methods
Problem Domain (P)
S:implement solution numerical methods
Visualization Domain (V)
V:display results based on specific simulation results
P:supplies problem data
PP:supply pre-processed data
S:Supplies simulation results
Pre-Processor Domain (PP)
Simulator Domain (S)
Simulation Domain (S)

Figure 3-5 Plexus Indicative Mood (Problem domains and their interaction)

The existing domains include:

- The Real World, where humans acquire expertise. It gives a clearer understanding of what the designer has to deal with;
- The Problem Domain, which includes problems to be solved through the simulation;
- The Simulator Domain, simulator main characteristics, and strategies, etc;
- The Pre-processor Domain, is represented by the input data mapped to the simulators requirements, and also some applied structures or machines used to do this mapping;
- The Simulation domain, which consists of simulation results;
- Visualization Domain, where more elaborate and appropriate results are generated and viewed.
- The User Domain is composed mainly of engineers and scientists.

Note that in Figure 3-5, we can identify some *problem frames phenomena*, which are part of the phenomena interface between existing domains. For example, between the User Domain and the Problem Domain we have an event controlled by the User Domain (U: Formulates, indicating that the User controls the problem formulation), etc. Those domains include a lot of information explained in Chapter 2 about FEM Simulation.

Next, we describe each of the involved domains. Some of them include existing commercial machines, which can be available in a FEM simulation analysis, such as commercial simulators (in the Simulator domain), mesh generators (in the Pre-processor domain). During this chapter, in the domains description we present meta-models, whose objective is to describe the involved concepts, syntax and semantic. Some of the meta-models represent the concepts that will help the formulation of the simulation problems and solution strategies in the FEM context. UML class diagrams will be used to represent the domains relationship. In particular, we define the following stereotypes: domain and phen-entity (which represents phenomena classified as entities).

3.3.2 Real World Domain

According to the context diagram, the Real World Domain is the scientific world where the engineer obtains information. In other words, it refers to the mathematical domain, where whole information about the problem to be simulated and respective solution methods come from. The mathematical world domain (Math World) is derived considering the physical world domain (Physic World), the geometry domain (Geometry World). The mathematical world domain (Math World), is in turn composed by the domain of exact mathematical world (Exact Math World), the domain of solution strategies (Solution Strategies World), and the domain of discretized mathematics (Discretized Math World). The FEM Domain is based on mathematical discretization of exact mathematical behaviour laws. The FEM Domain also restricts the Solution Strategies World, and is composed of Geometry Discretization and Phenomena Discretization sub-domains. The Real World Domain and the involved sub-domains are described in Figure 3-6.

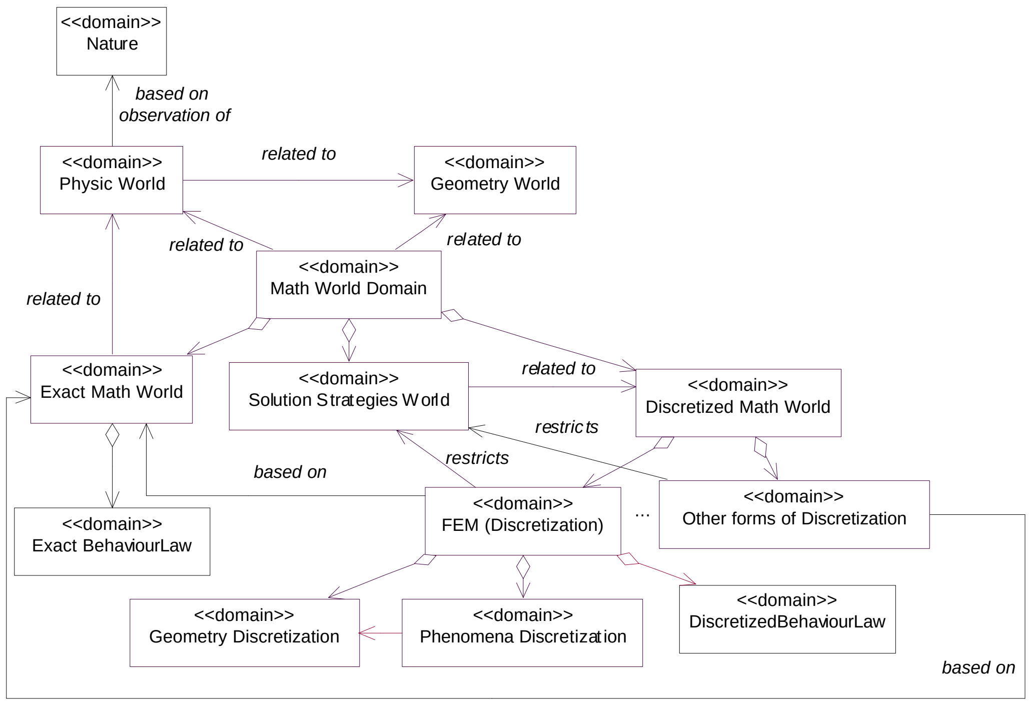

<<domain>> Nature
based on observation of
<<domain>> Physic World
related to
<<domain>> Geometry World
related to
related to
<<domain>> Math World Domain
related to
<<domain>> Exact Math World
<<domain>> Solution Strategies World
related to
<<domain>> Discretized Math World
restricts
restricts
based on
<<domain>> FEM (Discretization)
...
<<domain>> Other forms of Discretization
<<domain>> Exact BehaviourLaw
<<domain>> Geometry Discretization
<<domain>> Phenomena Discretization
<<domain>> DiscretizedBehaviourLaw
based on

Figure 3-6 Real World Domain

From Chapter 2, Figure 2-6 and Figure 2-7 we can identify several concepts, that is, *problem frames phenomena*, which compose the involved domains [JAC01]:

- Entities: Exact Vector Field (in Physical World), Exact Behaviour Laws (in Exact Math World), Exact Geometry (in Geometry World), Discretized Behaviour laws (in FEM), Geometric Mesh (in Geometry Discretization), Discrete Vector Field, Shape Functions (in Phenomena Discretization), etc
- Event: Discretization of Behaviour Laws (in FEM), Mesh Generation, Exact Behaviour Laws Formulation (in Exact Math World), and Physical Model Creation (in Physic World), etc.
- Values: Finite Element Geometric Shape = Triangle (in Geometry Discretization), and Mesh Generation Method = Restricted Delaunay (in Geometry Discretization), etc.
- States: Geometry Discretized or Mesh generated (in Geometry Discretization), Behaviour Laws Discretized (in Phenomena Discretization), and Solution Strategy Defined (in FEM Discretization), etc.
- Truth: Discrete Weak Form equivalent to the System of Algebraic Equations (in FEM domain), and Mesh Generation Method ShouldBeCompatibleWith finite element geometric shape (in Geometry Discretization), etc.
- Roles: Triangle defines Finite Element Shape in Mesh Generation (in Geometry Discretization), etc.

The domain phenomena (problem frames phenomena), that compose the domains in Figure 3-5, was be exemplified by a small set of concepts, because our purpose is only to illustrate how these definitions can help the understanding of the involved domain information (events, entities, truths and states, etc).

3.3.3 Problem Domain

The Problem Domain includes the exact and discretized problem:

- The exact problem consists of obtaining the exact vector fields, which satisfy the exact behaviour laws (coupled multi-physics) defined on a given exact geometry.
- The approximate (discrete or discretized) problem consists of obtaining the discrete vector fields defined on an approximated geometry, which satisfies the system of algebraic equations, which resulted from the discretization procedures applied to the exact behaviour laws.

In order to define a problem several pieces of data are needed and must be supplied by the User Domain. For instance: the exact geometry, phenomena data, phenomena-geometry relationships, phenomena-phenomena relationships, and auxiliary methods (e.g. mesh generation method), etc.

As seen in chapter 2, the discretization processes starts with the application of a FEM technique (from a FEM domain) the exact problems (in a Exact Math World domain), which will be transformed into the approximate (discretized or discrete) problem (from a FEM domain) by geometry discretization and phenomena discretization methods (from a Geometry Discretization domain and Phenomena Discretization domain, respectively).

Some of the *problem frames phenomena*, identified in the Problem Domain are, for example: (i) Entities such as the discretized geometry and discrete vector field. (ii) Events such as discretization; (iii) States such as DiscretizedProblem, which is a state of a given problem.

3.3.4 Pre-Processor Domain

Remembering that the definition of the discrete (or discretized) problem considers for granted several pieces of data. Some pieces are not usually given by the user and should be generated and their relationship with the discrete problem should be established before the simulation itself begins. For example, the geometric mesh. The “pre-processing” phase is responsible for the transformation of the user input problem data into data structures in the form acceptable by the analysis.

The Pre-processing domain includes *problem frames phenomena* such as: (i) the relationship between phenomenon and geometry entities, the relationship between phenomenon-phenomenon entities, and the definition of phenomena methods and processes; (ii) Events such as phenomena generation and geometry generation; mesh generation (geometric and phenomenon), and simulator configuration, etc.

The extent and complexity of the pre-processing phase depends on the level of abstraction supplied by the simulation system for the problem and solution definitions. Particularly, for coupled multi-physics problems, the pre-processing phase can become very intricate if high levels of abstractions are not considered.

3.3.5 Simulator Domain

This includes the computational system, which is used for the simulation itself, which means the computation of the solution to a given discrete problem. It executes processes such as: step estimation in the time progression scheme, model (physical and mathematical) and discretization adaptations, solution of systems of algebraic equations, and Error Estimation, etc. In spite of the commercial software available, which support FEM simulation, many problems are still beyond their scope. When defined/implemented from scratch, a simulator for complex problems demands hard work. Frequently simulators and simulations need to be redefined many times, which, without the definition of a standardized reusable way, implies heavy reprogramming. The main processes involved in a simulation were detailed in Chapter 2.

We can identify several concepts, that is, *problem frames phenomena*, which compose the Simulator domain. They include: (i) Entities such as: Pre-ProcessedProblem, SystemofEquations, and SimulationResults, etc; (ii) Events such as SimulateProblem, CalculatePhenomenaContributions, SolveSystem, and CalculateNextTimeStep, etc; (iii) States such as Error Estimated, and InitialTimeStepCalculated, etc; (iv) Values such as kindofsystem = ‘Linear’, Method = ‘NewtonCotes’, etc; (v) Roles such as SimulateProblem (Pre_processedData, and SimulationResults), etc.

3.3.6 Simulation Domain

This domain represents the entities that correspond to data generated during a simulation or the simulation result. It represents hard work, but frequently neither stored nor maintained. This domain must be verified and validated for the guarantee of proper results. So, some *problem frames phenomena* found in the Simulation domain includes: (i) Entities, such as Simulation Results and Pre-Processed Problems; (ii) States such as Problem Solved.

3.3.7 Visualization Domain

This domain corresponds to the “post-processing” that takes analysis output and generates the data required by the user in the form accepted by the viewer. The domain can include an appropriate machine (visualization environment), which helps this visualization; the physical problem solution (its result) is processed in order to obtain the quantities of interest for the user and for the required visualization. An important aspect is that the existing visualization environments require the data for visualization in a specific format.

Some *problem frames phenomena* found in the Visualization domain are: (i) Entities, such as Images, format, and visualization environment, etc; (ii) Events such as: View a specific Distribution (e.g. view the distribution of the involved stresses and temperatures in a structure), etc.

3.3.8 Users Domain

This domain includes persons that are responsible for events such as: building of simulators, running simulations and the visualisation of simulation results. They are composed of entities such as engineers, scientists and students, etc. That is, FEM simulator designers, developers and users, which are considered as *problem frames phenomena* entities.

In this section the knowledge domain (indicative mood) was described. We did not give many details because the FEM method has already been explained in Chapter 2. In the next section, the Plexus proposed environment (opative mood) is presented. The existing domains will be described together with some specific machines and given domains, which compose or interact with the Plexus machine.

3.4. Proposed Environment for the Development of FEM Simulators (Optative)

Our goal is to propose specific solutions, which can be gathered in a Simulation Environment for the development of FEM simulators (the Plexus System). Many applications are going to be developed, within the specific domain; so the timesaving of reuse will recover the time invested to develop it.

The desired Simulation Environment must improve domain comprehension (through abstractions and pre-defined data); support reusability of simulation models and numerical solutions; simplify requirements specification; focus on application specific functionality; and implement specialist routines for automatic programming, see Figure 3-7.

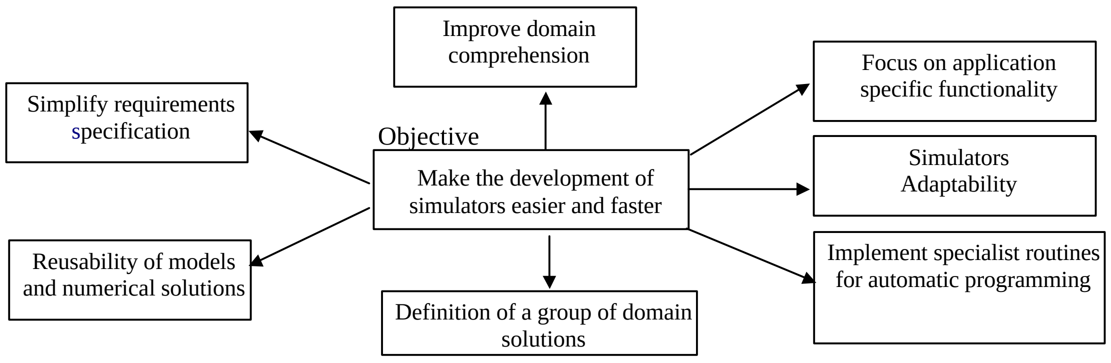

Improve domain comprehension
Simplify requirements specification
Objective
Focus on application specific functionality
Make the development of simulators easier and faster
Simulators Adaptability
Reusability of models and numerical solutions
Implement specialist routines for automatic programming
Definition of a group of domain solutions

Figure 3-7 Non-functional Requirements

The proposed environment also requires the management of great volumes of data, previously built components, phenomena, phenomena coupling, algorithms components, definition of persistent data and simulation knowledge reuse. There are also some systems, which may interact with the Plexus system, such as: (i) CAD systems, to input information related to geometries and so on; (ii) library components, since there are programs available that can be coupled to Plexus e.g. BLAS (Basic Linear Algebra Solvers); (iii) Image reconstruction systems; (iv) Visualization systems; (v) Virtual modelling systems.

The new Plexus environment must include features such as:

- Provision of a knowledge base management.
- Abstractions and pre-defined data for problem definition.
- A pre-processor machine, which helps the data treatment.
- Allow the use of different numerical methods in the simulation machine;
- Help simulator building and configuration, supporting the reusability of different simulators strategies.
- Support modularisation of simulator structuring.
- Definition of a systematic method to organize and describe processes in the FEM context in order to reuse them. FEM simulator process and process reuse, based on processes their types and levels of computation inside a FEM simulator. For processes to be reusable, we need to express common elements and variables within one process.

The general objective of our approach is to facilitate and shorten the development time of FEM simulators. To achieve this, the definition of a group of solutions for the FEM domain was considered: the definition of a specific domain architecture (applying abstractions which promote reuse), the definition of customisable and modular solutions, considering existing software engineering standards (i.e. reference models); management of commonality across different simulators; definition of adaptative models, and to explore process reuse. See Figure 3-8 .

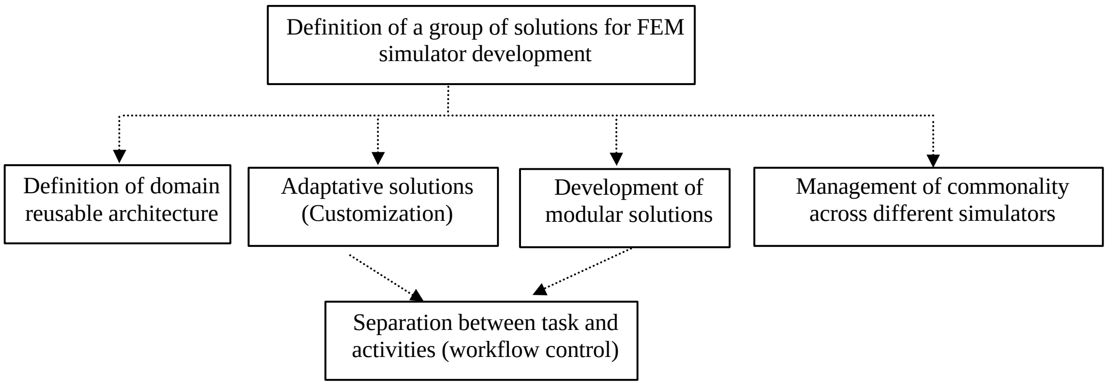

Definition of a group of solutions for FEM simulator development
Definition of domain reusable architecture
Adaptative solutions (Customization)
Development of modular solutions
Management of commonality across different simulators
Separation between task and activities (workflow control)

Figure 3-8 Solutions Decomposition

3.4.1 The Plexus System

The proposed environment will assist in the construction of simulators, based on meta-data information. It will support the development of various different classes of simulators, specified by the designer (engineer), not just one specific simulator. The developed simulator uses complex structures (that represent data and software components) to implement coupled phenomena simulators in the FEM context. The environment allows the design of an integral piece of software, which is able to solve the coupled phenomena problem as an integrated solution for the considered/defined coupling. Figure 3-9 gives an overview of the Plexus system input and output. This figure does not follows a specific notation, is only illustrative.

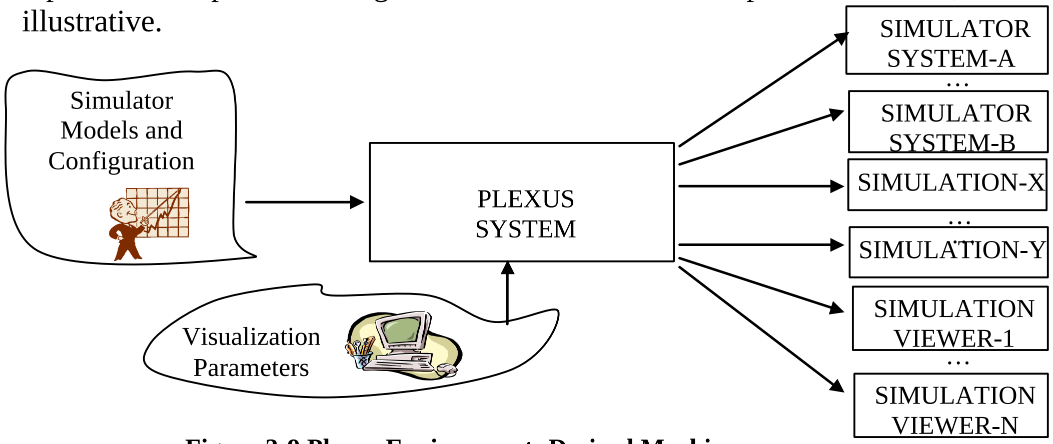


Figure 3-9 Plexus Environment -Desired Machine

In this work, however, we strive for the construction of reusable simulators, which take into account several requirements that will be further detailed. In Figure 3-10 we propose a general process for FEM simulation development, which involves the following activities: (i) Defining a meta-simulator model. (ii) Building the required simulator, that is, a semi-complete simulator based on the designer simulator model is constructed. (iii) Configuring this simulator, using defined articulation strategies[3], resulting in the final simulator. This simulator could also be reconfigured later. (iv) Using the final simulator for the definition of a general problem scenario. This, in turn, will run different requested simulations. Finally the visualization process can be used for a better understanding of the results.

In Figure 3-10, the correspondent states encapsulate the systematic similarities amongst simulators, allowing software FEM simulator developers to readily extend the framework components into applications that address specific simulator requirements. It provides a process organisation, which supports the required encapsulation for exploring reusability in the development of a new application. More details will be seen in the Plexus architecture, in Chapter 4. Hence, developers could write new applications with a proven design and assure the reuse of code and data.

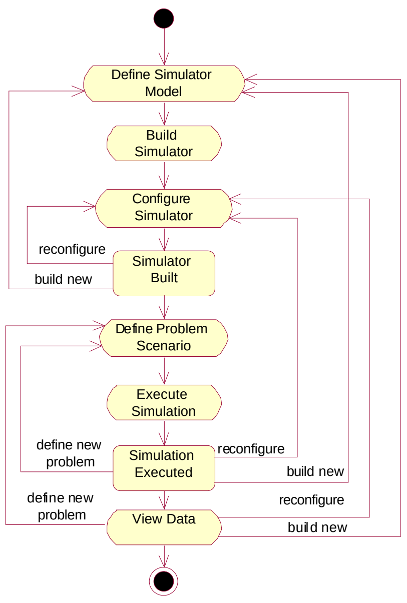


Figure 3-10 Simulator Building and Simulation -Problem Definition and Solution

An example of the application of the process is described as follows. Imagine that a specialist wants to build a simulator for the following scenario/requirements: a simulator capable of solving problems involving transient phenomena; the phenomena context includes temperature-dependent elasticity, rigid body motion

[3] The simulator main solution algorithm is here considered as an articulation of pre-defined processes. Hence, it occurs at a higher computational level then the "classical" simulator processes.

and heat transfer. First, the designer defines these models through the Plexus system. Then, the designer asks the Plexus system to build it. The Plexus identifies the proper global algorithm, taking into account the feature supplied and matches them with some pre-defined global algorithm skeletons, for the simulator and generates a semi-complete simulator. Then the designer configures the simulator with the phenomena *articulation strategies*, that is, identifying how phenomena will be grouped (their contexts) and solved in further simulated problems. As a result the simulator will be ready for the execution of simulations. At this stage, users of the simulator will define several scenarios for running simulations. In this work we will apply this approach. Two examples of problems that can be solved by the defined simulator are described in Appendix A. After or during execution of a simulation, the user can view data through a specific visualization tool.

Through the simulation environment the user is capable of decomposing the geometry into relevant components, where phenomena are defined. It is also possible to copy and selectively distribute geometry data between phenomena. The user can also define different solution strategies for different phenomena groups. Moreover, the user has the flexibility to define algorithms as data in several levels of the simulation, to integrate pre-built software components to the environment, and run local or remote simulations.

In the following subsection, we will consider simulator development problem decompositions (structuring) [JAC01].

3.4.2 The Plexus Context Diagram

The Plexus context diagram is presented in Figure 3-11. It is composed of the desired machine and the domains with which it interacts. Note that there are new domains, which were defined to give support to the new system, like Knowledge Base Domain and the Pre-Processed Data Domain, which were not present in Figure 3-4, and are part of the defined domains to give support to the machine. Also the User Domain was enriched with new users and now is called Plexus User Domain.

Some of the Plexus domains include sub-machines, which satisfy some of the defined requirements for the Plexus machine. They try to complement the automation of the existing world, described in Figure 3-5. These sub-machines are the ones described in Figure 3-24 and each one will take part of a specific domain.

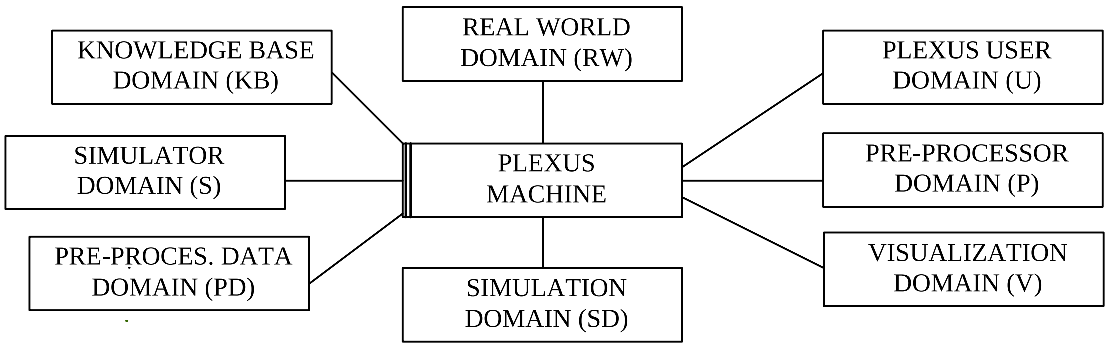


Figure 3-11 Plexus Context Diagram

Note that Figure 3-5 and Figure 3-11 represent different diagrams. The first presents a diagram of the indicative mood, while the second explains the machine to be built, together with the domains that will interact and support the required Plexus machine.

3.4.3 Real World Domain

This domain is the same as the one presented in the section 3.3. It is the outside world, which will supply information to users and to the knowledge (stored in the Plexus System).

3.4.4 Plexus User Domain

The Plexus User Domain (U) is composed of different users: Designer, Simulator User and Administrator, see
Figure 3-12. Remember that the Problem Frames represent people as biddable domains. The Designer Domain is, in turn, composed of scientists and engineers; they are responsible for the simulator model definition, or to supply the general information for the Knowledge Base Domain. The designers select information from the Knowledge Base Domain in order to specify a configurable simulator. The Simulator User Domain represents final system users, who will rely on the simulator for different problem solutions and running different simulations and post-processing. The persons that are responsible for the system configuration and table loading represent the Administrator domain; they are not necessarily FEM simulator specialists.

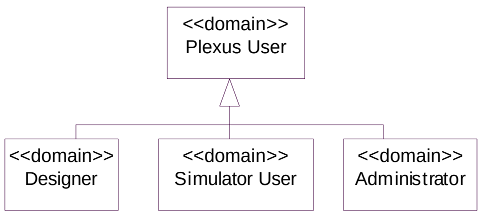


Figure 3-12 Designer, User and Administrator Domain

The Plexus Users Domain is intrinsically related to the Knowledge Base Domain.

3.4.5 Knowledge Base Domain

The Knowledge Base (KB) domain represents the discretized information (Geometry and Phenomenon Discretization domains, see Figure 3-6) that will take part of the system knowledge. The Knowledge Base domain is composed of: Basic Knowledge Domain, Simulator Knowledge Domain, and Problem Knowledge Domain. The Basic Knowledge data is the most reusable data; it helps Plexus users in the definition of simulators and simulation problems. The Simulator and Problem Knowledge domains are composed of data related to simulator and simulation problems respectively.

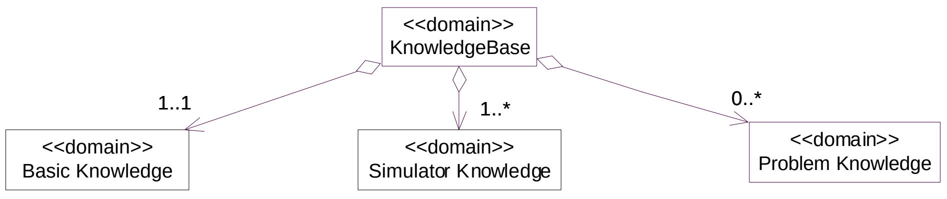


Figure 3-13 Knowledge Base Domain

In the sequence we present them in more detail.

a) Basic Knowledge Domain

This domain consists of the Geometry, Component and Phenomenon domains (see Figure 3-14).

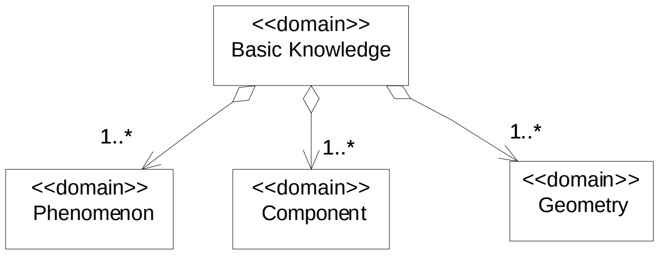


Figure 3-14 Basic Knowledge Domain

The Phenomenon Domain abstractly defines phenomena as a fact or occurrence, which can be described by a certain finite number of pieces of information, which, in turn, have to obey a set of behaviour laws (for example, fluid flows and heat transfer). This domain is composed of semi-defined phenomena, which will guide and further help simulation phenomena definition in the Problem Knowledge domain.

The Component Domain is an abstraction that represents software components, used for implementing parts of processes, e.g. simulator skeletons and phenomena methods. Hence, they represent the most reusable parts of the Plexus system.

The Geometry Domain consists of the supplied geometries. Each of the geometries is represented hierarchically[4], from the definition of the highest dimension geometric part, for instance - volume, down to the definition of the lowest dimension parts (i.e., points).

Other information that is part of the basic knowledge domain is related to existing possible classifications, such as: phenomena contexts, types of phenomena coupling, types of boundary conditions and restrictions.

b) Simulator Knowledge Domain

This domain is composed of the Algorithm Skeleton, Simulator Model and Configuration Domains, described below.

The Algorithm Skeletons Domain represents algorithms defined by a simulator designer, for a simulator. By *skeletons* we mean those parts of the solution process, which can be replaced, making it possible to build different simulation strategies (configuration).

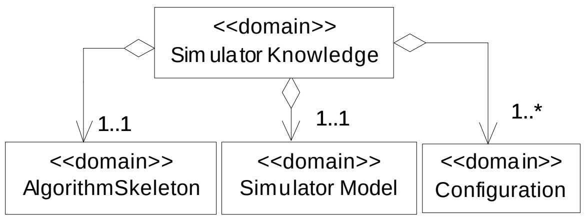


Figure 3-15 Simulator Knowledge Domain

The Simulator Model Domain includes the characterization of several simulators (see Figure 3-16). It is composed of:

- The Simulation Scenario determines the main features of simulation strategy: phenomena classes (transient, steady, etc.), estimation error (in space, time and model), adaptation (in space, time and model), etc. Each Simulation Scenario, also called Skeleton Specification, represents the classes of problems that a simulator will be able to tackle in a broad sense; it includes a list of simulator major features (type of phenomena, estimation error options and adaptation options).
- The Phenomena Context describes phenomena classification groups, e.g. heat transfer in solids or liquids, flow of Newtonian fluids, and linear or non-linear elasticity.

[4] **P**lexus uses the boundary representation method (B-rep) for geometry management.

- A Global Skeleton, which is an algorithm skeleton used to implement the highest level of the solution scheme of a simulator. It implements the simulator scenario.

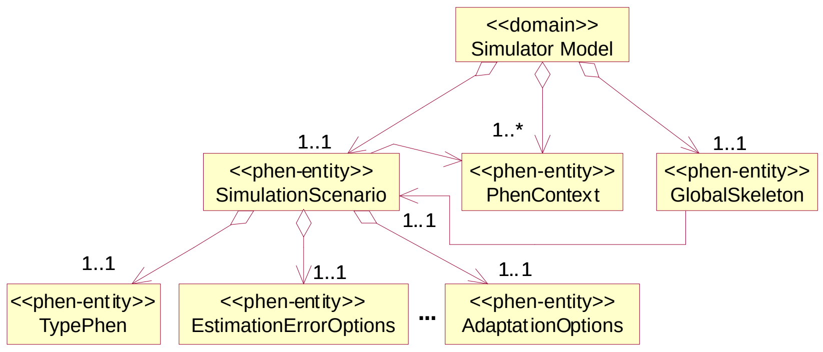


Figure 3-16 Simulator Model Domain

The Configuration Domain is related to simulator configurations. Each simulator configuration is represented by simulator articulation strategies, which describe the way involved phenomena will be solved together. Articulation strategies can be defined at block level (which consider different groups of phenomena) or group level (a group of phenomena); they are represented by Block Data and Group Data properties, described in Figure 3-17. More details, which justify the definition of blocks and groups of phenomena, can be seen in Chapter 5 (FEM-Simulator Skeleton Pattern).

Block Data is defined by:

- Block context, which represents the general classification of the phenomena grouping. We can have, for example, the heat transfer and elasticity block contexts. This means that in a specific problem we can have the heat transfer problem solved as one block, while another block solves the elasticity problem. Each of these blocks involves groups of phenomena compatible with its context.
- Block Skeleton, which is an algorithm related to the block level of computation. Represents processes involving groups of phenomena in the simulation.
- Block Method, which is a method (another skeleton) used by a block.

Group Data is defined by:

- Group Context, which is the general classification of phenomena grouping, considering the phenomena contexts.
- Group Skeleton, which is an algorithm related to the process in the group level of computation, representing strategies for solving groups of phenomena;
- Group Method, which is a method (another skeleton) used by a group.

Examples of Block and Group Skeletons and Context can be found in the Chapter 5, which details a FEM-Skeleton pattern.

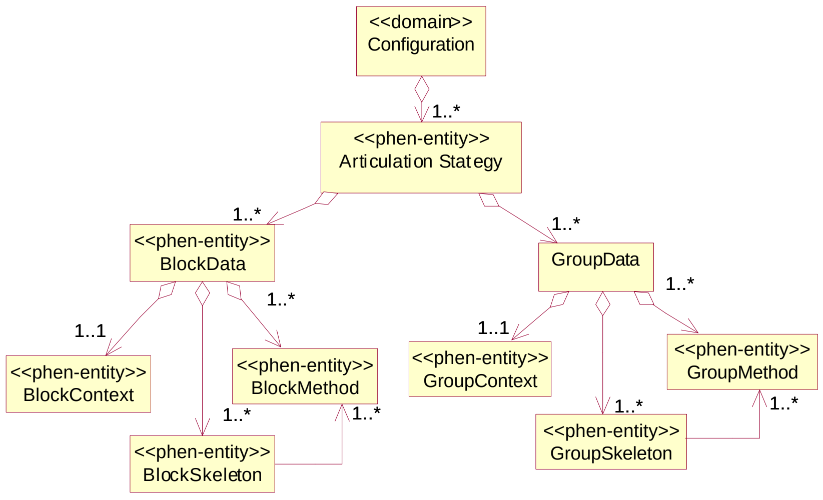


Figure 3-17 Configuration Domain

So far we have described the Basic and Simulator Knowledge Domains (according to Figure 3-13). Now we will describe the Problem Knowledge Domain that is related to problem data, which will be solved by the defined simulators.

c) Problem Knowledge Domain

This domain consists of several simulation problems to be solved by a simulator. Each problem is composed of: the geometry where the problem takes place, phenomena under study, simulation regions, representing the geometry entities where these phenomena occur and blocks and groups of phenomena (see Figure 3-18). By group, we mean a set of phenomena, which are going to be solved monolithically (together). By block, we mean a set of groups of phenomena, which will be articulated in a solution branch independently from other blocks. The definition of more than one block is justified in the case where a problem can be partitioned into either independent or coupled sets of groups of phenomena. For each group of phenomena the user must specify the scenario (front tracking, type of linear solver, equation type, etc) that will generate specific algorithm skeletons and constrain the group methods.

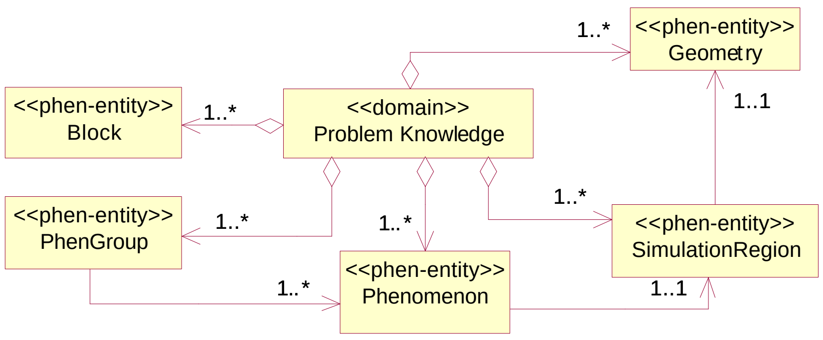

<<phen-entity>> Geometry
1..*
<<phen-entity>> Block
1..*
<<domain>> Problem Knowledge
1..1
<<phen-entity>> PhenGroup
1..*
1..*
1..*
<<phen-entity>> SimulationRegion
<<phen-entity>> Phenomenon
1..*
1..1

Figure 3-18 Problem Knowledge Domain

Phenomenon is composed of complex abstraction tools responsible for providing the contributions of a natural phenomenon to the algebraic equations to be solved in each instant of the solution process.

Having described the Knowledge Base Domain (KB) we can now move on describe the Simulator Domain (S).

3.4.6 Simulator Domain

The Simulator Domain consists of (see Figure 3-19):

- The Simulator Builder, which corresponds to the software program and data used to construct the simulator.
- The Simulator Configurator.
- Simulation Data corresponds to the pre-processes data supplied by the Pre-processor.
- The Simulator Kernel, which is a workflow that executes a simulator specification, applying the Simulation Data.
- The Simulator Domain is described in sequence.

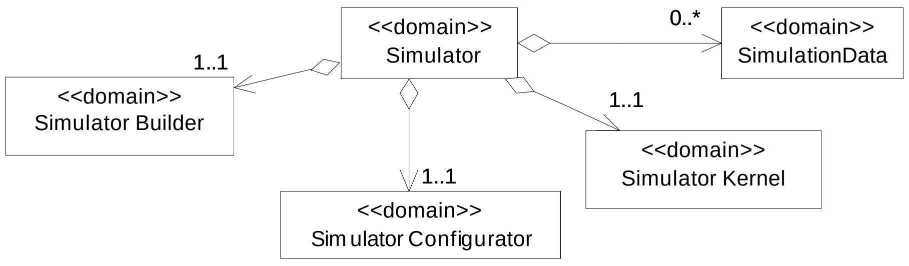

<<domain>> Simulator
0..*
<<domain>> SimulationData
1..1
<<domain>> Simulator Builder
1..1
<<domain>> Simulator Kernel
1..1
<<domain>> Simulator Configurator

Figure 3-19 Simulator Domain

The Simulator Builder Domain is composed of a machine (tool) and data, which are capable of building a semi-complete simulator considering a defined or selected simulator model. A Semi-complete simulator represents a framework, which is responsible for the control of the main process flow (Simulator Kernel), i.e. the user's simulation strategy implementation, which will guide the remaining processes (pre-processing, simulation, and post-processing). It is built from a series of decisions about the kind of problem that the strategy will satisfy, for instance the type of phenomena (transient, pseudo-transient), error estimation and type of adaptation (if it exists). Hence, it represents the basic infrastructure for simulation execution. It is the class that can be customized with the possible simulations, which represent the main process. Thus, it maintains the core of simulation through the global algorithm skeleton.

The Simulator Configurator Domain represents the machine (tool) and data for configuring a simulator. A configured simulator means a simulator state where data about the desired articulation strategy has already been supplied for the semi-complete simulator. By articulation strategies we mean: simulator configuration data following the specified simulation scenario, which determines the block and group skeletons, the number and type of blocks and groups of phenomena, their skeletons and relationship and execution order.

The Simulator Kernel Domain is the main process (engine) that executes the simulator workflow. Figure 3-20 presents a defined workflow process to be controlled. In Plexus, the main skeleton defined for a simulator represents the workflow business process. The invoked applications include the Block and Group Skeletons, and the Computational Phenomenon (which is the complex abstraction tool responsible for providing the contributions of a natural phenomenon to the equations to be solved in each instant of the simulation solution process). The Kernel is represented by the Simulator component detailed in Chapter 4.

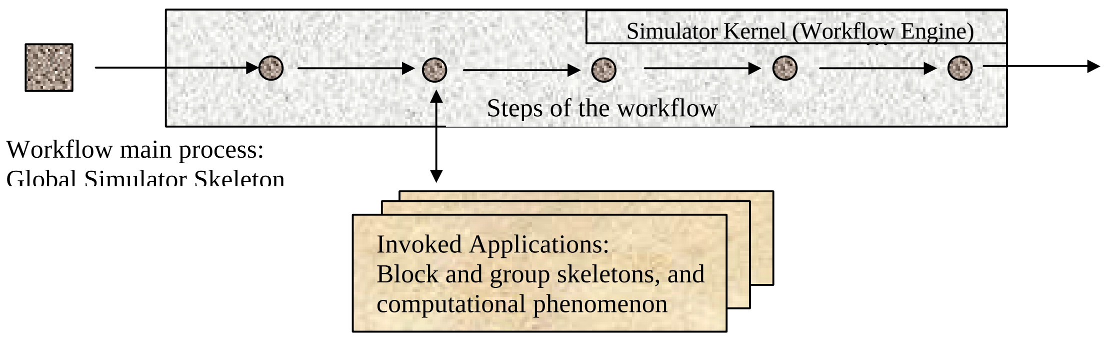
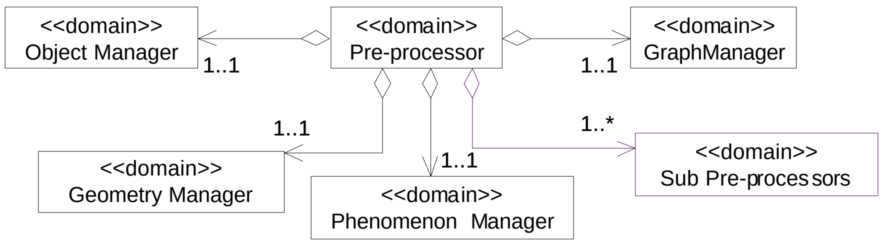

Simulator Kernel (Workflow Engine)
Steps of the workflow
Workflow main process:
Global Simulator Skeleton
Invoked Applications:
Block and group skeletons, and
computational phenomenon

Figure 3-20 Business Process Workflow controlled by the Simulator Kernel

Simulation Data Domain is composed of the Problems Data - Phenomena, Geometry, and Finite Element Domains - after the appropriate discretization.

3.4.7 Pre-processor Domain

The Pre-processor Domain includes a machine (a tool) that will map the users input data (Problem Data) to the suitable Pre-processed Data structure (Figure 3-21). It is independent of the simulator itself, and is related only to the data structure used by the simulator.



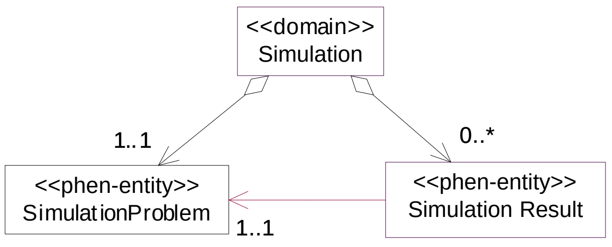
<<domain>> Object Manager
<<domain>> Pre-processor
<<domain>> GraphManager
1..1
1..1
1..1
1..*
<<domain>> Geometry Manager
1..1
<<domain>> Phenomenon Manager
<<domain>> Sub Pre-processors

Figure 3-21 Pre-processor Domain

The Pre-Processor Domain is composed of specific managers, which deal with objects used in the generic pre-processor control (Object Manager Domain), geometry (Geometry Manager Domain) and phenomena (Phenomena Manager Domain), see Figure 3-21. It also includes sub pre-processors, which deal with specific data mappings. More details about these tools are given in chapter 4, the pre-processor subsystem section.

3.4.8 Simulation Domain

The Simulation Domain is composed of the following domains: the first is the Simulation Problem, which corresponds to the problem which will be solved by the simulator and that will be mapped to the appropriate Simulation Data, after the pre-processing. The second is the Simulation Result Domain, which corresponds to the data obtained after a simulation run (see Figure 3-22).



<<domain>> Simulation
1..1
0..*
<<phen-entity>> SimulationProblem
<<phen-entity>> Simulation Result
1..1

Figure 3-22 Simulation Domain

3.4.9 Visualization Domain

The visualization domain is composed of several domains described bellow and presented in Figure 3-23.

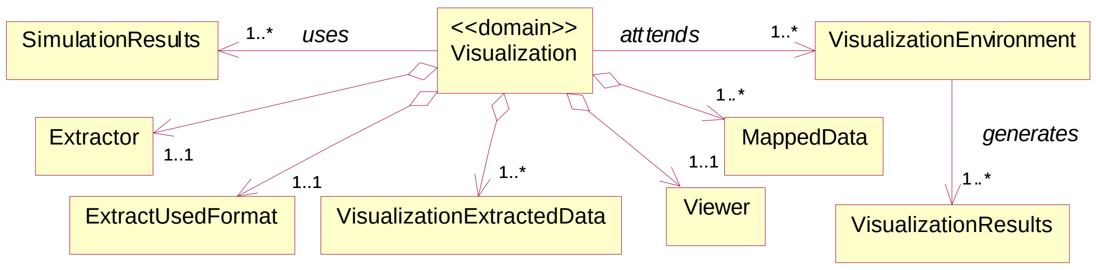


Figure 3-23 Visualization Domain

- Simulation results are the data generated for a problem simulation.
- An Extractor selects, gather and maintain relevant data to be used for visualization (the VisualizationExtractedData).
- ExtractUsedFormat is the specific data format used by the Extractor to maintain the extracted data.
- The Viewer is responsible for mapping the extracted data to a format that can be understood by an existing Visualization Environment.
- The data in the appropriate format (known by the visualization environment), generated by the Viewer is called the MappedData.
- Visualization Environment (the applications available in the market, e.g. Data Explorer [IBM02] and AVS [AVS02]).
- Visualization Results (which are relative to the final data presented by the used environment).

3.4.10 Classification of Plexus concepts according to Problem Frames

Remember that following [JAC01], the domains existing concepts are *problem frames phenomena*. They can be classified into: entities, events, values; states, truth, relations and roles. This classification helps to understand Plexus's concepts:

- There are some basic *entities* such as Simulator, Phenomenon, Geometry, Groups of Phenomena, Algorithm skeleton, Group of Phenomena and Numerical Methods.
- There are some defined *values*, such as: (i) types of phenomena coupling such as: weak, strong, etc; (ii) phenomena contexts, for instance, heat transfer in solids or liquids, flow of Newtonian fluids, Linear elasticity or non-linear elasticity. Boundary Conditions Types (which can be for example: Dirichlet, Neuman and Mixed)
- A Simulator can have the following *states*: (i) modelled simulator, which indicates that a specification of a simulator using a pre-defined model has been developed; (ii) built simulator, indicates when the simulator code has already been built; (iii) configured simulator, used when data about the desired articulation strategy have already been supplied for the simulator; (iv) simulation done, after a simulation has been finished.
- Existing *events* are for example: (i) input simulator model, (ii) input articulation strategies, (iii) build simulator, (iv) start simulation running, (v)

change simulator configuration, alter block and group articulation; (vi) input knowledge data; (vii) input problem data.
- Event *roles*, that is, the participation of one or more individuals in an event: BuildSimulator (Simulator_x, SimulatorM1l), Configuration (Simulator_X, Articulation_Strategies_Y); SimulationRun (Simulator_X, Preprocessor_K, ProblemScenario_Y, Results).

Until now we have described the Plexus context diagram, the involved domains and phenomena. In the sequence, for a complete analyse of the problem and specification, we will introduce the Plexus Problem Decomposition. The context diagram influences this phase, since the identification of the system required problem frames takes part of the problem decomposition.

3.4.11 Plexus Machine Structuring

The Plexus approach for the construction of simulators consists of sub-problems. Figure 3-24 shows the Plexus machine, which is composed by other sub-machines. This decomposition was based on the previous described context diagram and other information related to the proposed solution (e.g. the requirements of knowledge base management and simulator configuration):

- Knowledge Base Manager: is responsible for the maintenance of all information to be built and reused by the simulator.
- Simulator Builder: construction of a simulator based on a selected global scenario. Its concern is to ensure the right construction of a semi-complete simulator (based on the defined simulator model) responsible for simulator functioning. It ensures that the designer can further re-configure it, supplying the desired articulation strategies.
- Simulator Configurator: redefines simulator configuration from some designer supplied data. It changes the built simulator articulation strategies (see configuration domain for more details) by selecting other ones; and allows the designer to change the relationship between phenomena grouping.
- Pre-processor: responsible for the mapping/transformation of the user input data into a specific and appropriate defined data structure, which will be used by the simulator. It is described in more detail further on this chapter.
- Simulator: corresponds to the machine, which will be able to execute several desired simulations. It can be considered as a simple workflow structure that executes the constructed and configured Simulator, using Simulation Data, which includes Computational Phenomena (see GIG-pattern in Chapter 5).
- Viewer: allows further post-processing of simulations result data.

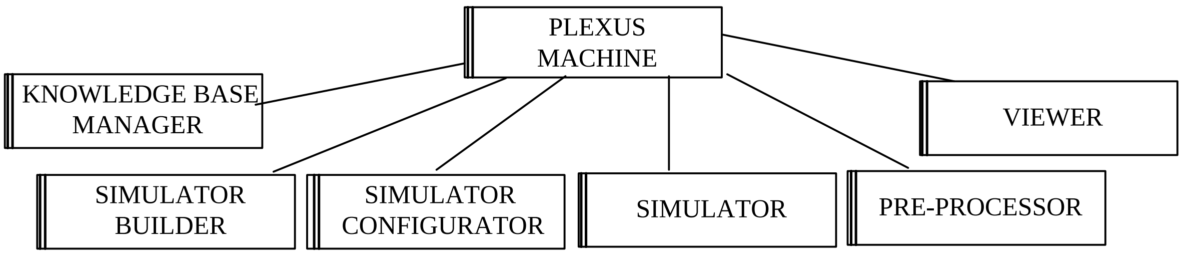


Figure 3-24 Problem's decomposition – Plexus Sub-machines

This decomposition was based on the requirement for building an environment for the development of simulators, following FEM conventional analysis approach. Next we will detail the main Plexus machine and each involved domain. Since the main Plexus problem frame is very complex, it must be decomposed into several simpler problem frames. In order to give an illustration of the representation of a class of sub-problems, we will detail the Simulator Builder Problem Frame and the Pre-processor Problem Frame in section 3.6.

3.5. The Plexus Problem Frames

The identification of a set of problem frames, specialized in FEM simulator development, will assist developers in system description and analysis. To illustrate the Plexus problem frames we choose to describe the Simulator Builder and the Pre-processor Problem Frames. Each decomposed sub-problem has its own partial views of the world and machine, taken from the original problem and is shown in a *problem diagram.* As described in section 3.2, a *problem diagram* has a concern, is composed by a machine, parts of the world with which the machine interact (involved domains which can be given or designed), the requirement description and the interface of shared phenomena (the specification phenomena) and the requirement phenomena (a predicate over the phenomena).

In a complete description, the defined Plexus Problem Frames could be further decomposed into instances of Jackson's basic problem frames [JAC01]. For example, the Knowledge Base Generator frame can be further decomposed into simple work-piece frames and the Pre-processor and the Simulator Builder into transformation frames. This identification helps in further descriptions, once Jackson details the way of describing each of them.

3.5.1 Simulator Builder Problem Frame

This problem frame concern is to ensure the construction of a simulator from a selected global scenario with a default configuration, Figure 3-25. The Simulator Builder finds the information related to the stored simulator model specified by the designer (through the global scenario and global skeleton). It constructs the simulator considering the default configuration (group and block skeletons) supplied by the knowledge base. The process of building the simulator is mainly concerned with the integration of the global, block and group skeletons defined by the configuration data.

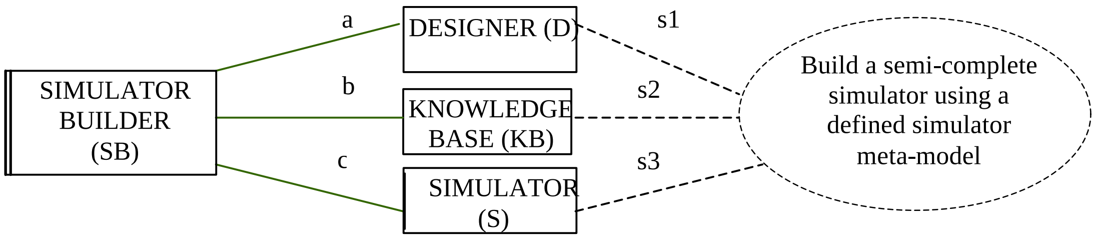


Figure 3-25 Simulator Builder Problem Frame

In order to simplify the diagram in Figure 3-26, a decision was made to use letters (a,b,c,..) in the specification phenomena, instead of including the controlling domains which will be presented next, following [JAC01] notation. The interfaced phenomena and the domains that control them are represented in the diagram (the Simulator Builder machine and the other involved domains: Designer, Knowledge Base and Simulator).

The requirement specification, in a problem frame, includes the indicative mode (what exists independently of the machine defined) and the optative mood (the requirements of the defined machine). The domains the machine interacts are represented by the indicative mode (see Figure 3-6) (Knowledge base, Designer and the Simulator). The optative mood is represented by the specification phenomena (see Figure 3-27) and the requirement phenomena (see Figure 3-28). Note that, in one problem frame the optative mood can be the indicative mood and in the other not. E.g. the Knowledge Base, which is optative in the Knowledge Generator is indicative in the Simulator Builder.

Problem Domain (Indicative Mood)

The Simulator Builder Machine interacts with the following domains:

- The Designer, is a biddable domain, which selects simulator global scenarios and algorithm skeletons and starts simulator building;
- The Knowledge Base Domain is a lexical domain, which supplies information about the simulator model;
- The Simulator is the built domain that is a causal domain.

Figure 3-26 Indicative Mood – Simulator Builder Problem Domain

Note that the machine Simulator Builder (SB) controls some actions:

- It interacts with the designer to get the required simulator model, and validates input data.
- It asks the knowledge base for the simulator model data, that is: the simulator scenario and algorithm skeletons, etc.
- It sets the default configuration.
- It builds the simulator.

Figure 3-27, details some examples of the interfaced phenomena and which domain is controlling them. A list of phenomena is associated with each controller domain, and describes the phenomena it controls. The symbol ! separates the list of controlled phenomena from the controller domain. The controller domain is identified by an abbreviation using uppercase letters (like D and SB). For example D!{Select_Simulator_Model} means that the designer selects a specific simulator. In turn, the Builder gets this information from the Knowledge Base (SB!{Request_Simulator_Model}).

Specification phenomena (Optative Mood)

The simulator builder "requests the Simulator Model" and the designer "selects, through the available ones":
a: SB! {Request_Simulator_Model}
D! {Select_Simulator_Model, Start_Simulator_Building}
The simulator Builder requests from the Knowledge Base the respective simulator data, which answers with the requested data
b: SB! {Request_Simulator_Data}
KB!{Give_Global_Skeleton_Data, Give_Default_Configuration }
The simulator Builder, constructs the simulator with a default configuration:
c: SB! {Set_Default_Configuration, Build_Simulator}

Figure 3-27 Optative Mood: Simulator Builder Specification Phenomena

Requirement phenomena (Optative Mood)

The system requires that:
Someone selects a pre-defined simulator model and start the simulator building:
s1: {Simulator_Model_Selected, Simulator_Building_Requested}
The existing selected model be retrieved from the knowledge base:
s2: {Simulator_Model_Given}
A simulator is built, if everything is ok:
s3: {Simulator_Built}

Figure 3-28 Optative Mood: Simulator Builder Requirement Phenomena

3.5.2 Pre-processor Problem Frame

This problem frame is concerned with the processing and constructing of dynamic structures (called pre-processor control structures) that will generate suitable simulator internal data, from the problems input data, see Figure 3-29. This data is related to mathematical and physical information (e.g. geometry, boundary conditions and vector fields). The system required states are presented in Figure 3-31.

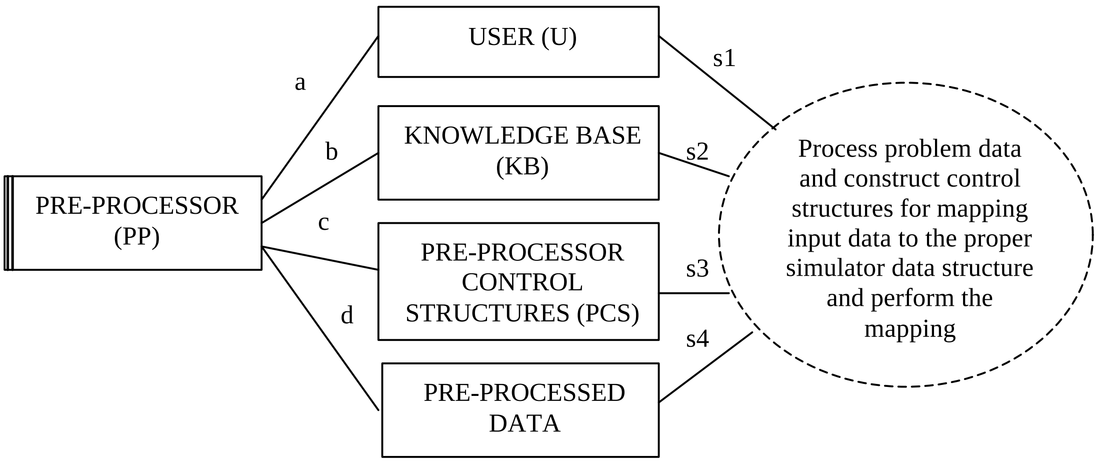

USER (U)
KNOWLEDGE BASE (KB)
PRE-PROCESSOR (PP)
PRE-PROCESSOR CONTROL STRUCTURES (PCS)
PRE-PROCESSED DATA
a
b
c
d
s1
s2
s3
s4
Process problem data and construct control structures for mapping input data to the proper simulator data structure and perform the mapping

Figure 3-29 Problem Frame Pre-processing

The Pre-processor machine interacts with the following domains, see Figure 3-30: Knowledge Base (KB), which stores existing reusable data (e.g. problem data); User (U), a biddable domain which desires to solve simulation problems; Pre-processed Data domain (PD), which is a given lexical domain, which represents the data structure that will be used further in the simulation. It also has to interact with the Pre-processor Control Structure Domain (PCS), which is a causal domain that represents appropriate controllers for the input data mapping of the simulator data structures.

Problem Domain (Indicative Mood)

The Pre-processor Machine interacts with the following domains:

- User (U);
- Knowledge Base Domain (KB),
- Pre-Processor Control Structures (PCS)
- Pre-Processed Data (PD)

Figure 3-30 Indicative Mood: Pre-Processor Problem Domains

Figure 3-31 describes the desired states to be achieved by the Pre-Processor machine, such as Pre-processor control structures are mounted.

Requirement phenomena, that is desired states (Optative Mood)

The system requires that:

s1: A problem is identified, and the pre-processor started

s2: Problem data is retrieved

s3: Pre-processor control structures are mounted

s4: Simulator data is generated

Figure 3-31 Optative Mood: Pre-processing Requirement Phenomena

Note in Figure 3-32, that the Pre-processor machine (PP) controls events such as: Geometry Pre-processing, Phenomena Pre-processing, Group and Block Pre-processing, Block Pre-processing, Geometry Mesh Generation, Phenomena Mesh Generation, Geometry Copying and Collapse and so on. Each one can be decomposed into a problem frame.

Specification phenomena, that is some of the possible events

The pre-processor "requests the Problem Scenario" and the user "selects, through the available ones":

a:PP! {Request_Problem_Scenario}
U! {Identify_Problem_Scenario, Start_Pre-processing}

The Pre-processor requests from the Knowledge Base the respective problem data, which replies with the requested data

b:PP! {Request_ProblemData}
KB!{Gives_Problem_Data (Simulation_Regions, Phenomena,…)}

The Pre-processor, runs several sub-pre-processes and generates the simulator data

c: PP!{PP!{PreprocessGeometry, PreprocessPhenomena, PreprocessGroup, etc}
d: PP! {Generate_Simulator_Data}

Figure 3-32 Optative Mood: Pre-processing Specification Phenomena

We can identify that the Pre-processor includes one of Jackson's basic Transformation problem frames [JAC01], which describes a machine where some readable input files data is transformed to a certain output file (with a particular format), according to certain rules. In the pre-processor we can identify: some input domains (Knowledge Base and Designer), output domains (Pre-processor Control Structures and Simulator Data) and requirements. Although Pre-processor control structures are output, they are also internal data, which will be further transformed to generate the output simulator Data.

3.6. Problem Frames Evaluation

In this chapter, we choose the Problem Frames technique for our complex engineering domain description due to several aspects. Some were related to the separation of the problem domain and the machines (indicative and optative moods), others due to the use of Problem Frame diagrams to represent particular solutions of classes of problems. In this work, we described a sequence of steps for FEM simulator structuring that promotes reuse of components and data. Some advantages and problems are enumerated below (establishing the benefits, and suggesting that the approach is valuable, however not complete).

The complex domain of FEM involves different sub-domains. Engineers need a technique, which uses common and compatible terms with their scientific/engineer's terminology. As FEM is complex, simplicity in modelling is also a desire. The extensibility and reuse of the previous solutions helps to reduce development time, and improves the level of correctness. It is also important to represent the rationale and design decisions for the requirements, which are not considered in this chapter.

We can make four general observations about the advantages of the use of the Problem Frames technique [JAC01] in FEM simulators:

- First, related to the appropriateness of the provided terms: the term domain is commonly used in mathematics, and also the term machine is applicable to simulator development. However, we have problems with the term phenomena, which cause confusion, since it has different meanings in Plexus (where we consider physical natural phenomena) and in the Problem Frames approach (where it means anything in the world). Also the term domain can be seen as very strong.
- Second, regarding reuse. Problem frames are one of a number of ways of high level grouping problems by type. Problem frames are similar to design patterns - elements of reusable object-oriented software, but are problem oriented rather than solution oriented. Problem frames make it easier to solve a problem once the type of problem is classified. In FEM, we use problem frames to capture an abstraction of a class of problems, for example the construction, configuration and pre-processing of simulators. Hence, it allows the definition of problem frames to be reused and detailed in a standardized way. This is reinforced by the fact that FEM is a domain specific application, where we can take advantage of existing expertise such as the reuse of skeletons.
- Third, the levels of detail are very appropriate. Problem frames have a distinctive characteristic, which is very attractive: they clearly separate the indicative and optative mood. In FEM this feature is used to describe the existing problem in a modularised way identifying the definition of a simulator exploring a reusable form through meta-models. Also the concept of phenomena, which involves the concepts of entities, states, and events, allows a clear enumeration of the involved data, and operations, and so on. In FEM we have for example the following states: Simulator Modelled, Constructed and Configured. This simplifies cross-references in the model definition. Another relevant aspect is the definition of the context diagram at problem level, giving an overview of the whole problem to be considered. Furthermore, Problem Frames allow the description of their involved domains using any language, such as UML, which is used in the Plexus meta-model definition.
- Fourth, some other gains are generic to requirements engineering, such as the support for precise definition of concepts through a more formal description (not discussed in this work), support for modularity in the definition of architecture in early stages, that helps system analysis and evolution.

We also identified some disadvantages in Jackson's technique [JAC01]. Despite the appropriateness of some terms, there are still some ambiguities in the way the concepts and categories can be applied. The definition of a meta-model may help to address those ambiguities or misunderstandings. The problem frames notation is neither simple, nor intuitive, nor trivial. The complete documentation becomes very long. The need for predefined abstractions for representing the relationship between domains (like composition and restriction) could help domain grouping, comprehension and definition. It also does not identify some relevant notions present for example by Bubenko [BUB95] (e.g. causes and problems, applied in Chapter 2 for characterising an existing problem). Finally, Problem Frames do not treat goals explicitly and neither assists in solving conflicts.

3.7. Final Considerations

In this chapter Problem Frames were applied to structure the analysis of the world in which the problem is located and describe the involved concepts and what effects one would like the Plexus system to achieve.

Next chapter will detail the definition of a specific architecture for the Plexus Simulator Environment, which will manage commonality across different simulators. The architecture is defined considering some quality attributes to which it maintains conformity (such as system reusability, flexibility and adaptability, as well as interaction with other systems).

Chapter

4

Plexus Simulation Environment Architecture

This chapter presents the Plexus architecture for supporting simulation of coupled phenomena based on FEM solutions. This architecture takes into account the required quality attributes, architectural components, and the interaction between components and their functionalities. The chapter gives a clear perspective of the whole system and the control required for its development, aiming to reflect system requirements such as reuse, modularity, and flexibility. The architectural abstractions used include frameworks and patterns, which are detailed in the next chapter.

4.1 Introduction

Understanding of a domain feeds requirements that help to define an architecture, which determines components. In chapter 3 we made an in-depth analysis of our problem domain, FEM simulator development.The requirements for the proposed environment, called Plexus, were pointed out and discussed. In addition, the first structuring of the Plexus environment was proposed. In this chapter, we suggest an architecture and its components for this environment, considering the current trends in software engineering development. The approach takes into account the definition of a structure that improves the quality of the simulator designs. The defined architecture attempts to fill the existing gap in the development of FEM simulators.

Some of the previous identified requirements are drivers for the Plexus's architecture, such as: flexibility in the development of simulators, extensibility of the system through component integration, reduction of complexity, reusability (processes, data and models), and distribution. Quality attributes, such as *performance* are not the focus of this work, since many other studies are being developed in this area, such as performance analysis in the solution of linear and non-linear systems, and performance in matrix and vector calculations. However, this work tries to guarantee *flexibility in the integration of the achieved results* obtained in those performance research areas through the integration and modularity quality attributes of the system. Furthermore, the requirement of being a cooperative system was not considered in this work, due to the complexity involved in addressing it.

Architecture is a design artefact that begins to map requirements into a solution. Quality attributes of a system are mainly permitted or precluded in its architecture. Hence, if the architecture does not comply with these qualities, from the beginning, one cannot expect to achieve them later on in the development. The architecture determines the structure and management of the project development as well as the resulting system, since teams are formed and resources allocated around architectural components. For anyone seeking to learn how the system works, the architecture is the place where understanding can be improved [CNL01].

The system architecture concept represents the main support for the development and maintenance of software systems of long term. One interpretation of software architecture definition is that presented in [BCK98]: "It is the structure or structures of the system, which comprises software components, the externally visible properties of those components, and the relationship among them". *Externally visible properties*, refers to the assumptions that other components can make of a component, such as its provided services and shared resource usage. By making *externally visible properties* of components part of the definition, one intentionally and explicitly includes component interfaces and their behaviours. Components may be developed by the internal team, bought on the open market, mined from legacy assets, or commissioned under contract. Once available, the components may be integrated to the system and tested [CNL01].

Software architecture can be observed better in terms of views. Views are particularly useful as guidelines for implementing and maintaining a system. Subsystems and components are typically specified into different views to show the relevant functional and non-functional properties of a software system [BUS96, BCK98]. Views in UML (*Unified Modelling Language* [OMG02]) can capture the structural and behavioural aspects [HR99]. Classes and packages can represent a structural view; on the other hand, the behavioural view can be represented by scenario, states and activities.

As a generic modelling language, the UML offers a familiar notation for designers, besides allowing a direct link between object-oriented implementation and development tools. However, as a general-purpose language it has the problem of conceptual object vocabulary that cannot be ideal to represent architectural concepts. As there is a considerable interest in using general notation for architecture modelling, a great number of proposals have recently attempted to show how concepts found in ADLs (Architecture Definition Languages) can be directly mapped into an object-oriented notation such as UML [ME99, HNS99, GKP99]. However, the purpose of this work is to give a generic view of Plexus architecture, not considering the details found in ADLs (such as ports and interfaces).

The next section describes Plexus architecture subsystems and components. Plexus architecture will be presented according to the conventional UML approach, that is, the definition of the structural and behavioural views. In section 4.2, the Structural View Section shows the architectural components and their functionalities. After, in section 4.3 the Behavioural View Section describes component interaction. Then in section 4.4, the defined architecture is evaluated, considering the addressed quality attributes.

4.2 Plexus Architectural Structure

The Plexus system is described by three-levels of architecture, which improves the portability and maintainability of the system. The main levels are the interface, which corresponds to the presentation, logical, and repository levels. The general organization of the system and global control structure are presented in Figure 4-1 using UML package notation:

- The presentation level is composed of the Plexus interface, which is responsible for all human-system interaction and visualization of results.
- The logical level representation includes the following major subsystem: Knowledge Management, Pre-processor, Simulator Builder, Simulator Configurator, Pre-Processed Problem Data, and Simulator.
- The data storage level is composed of a single subsystem called the Repository Manager.

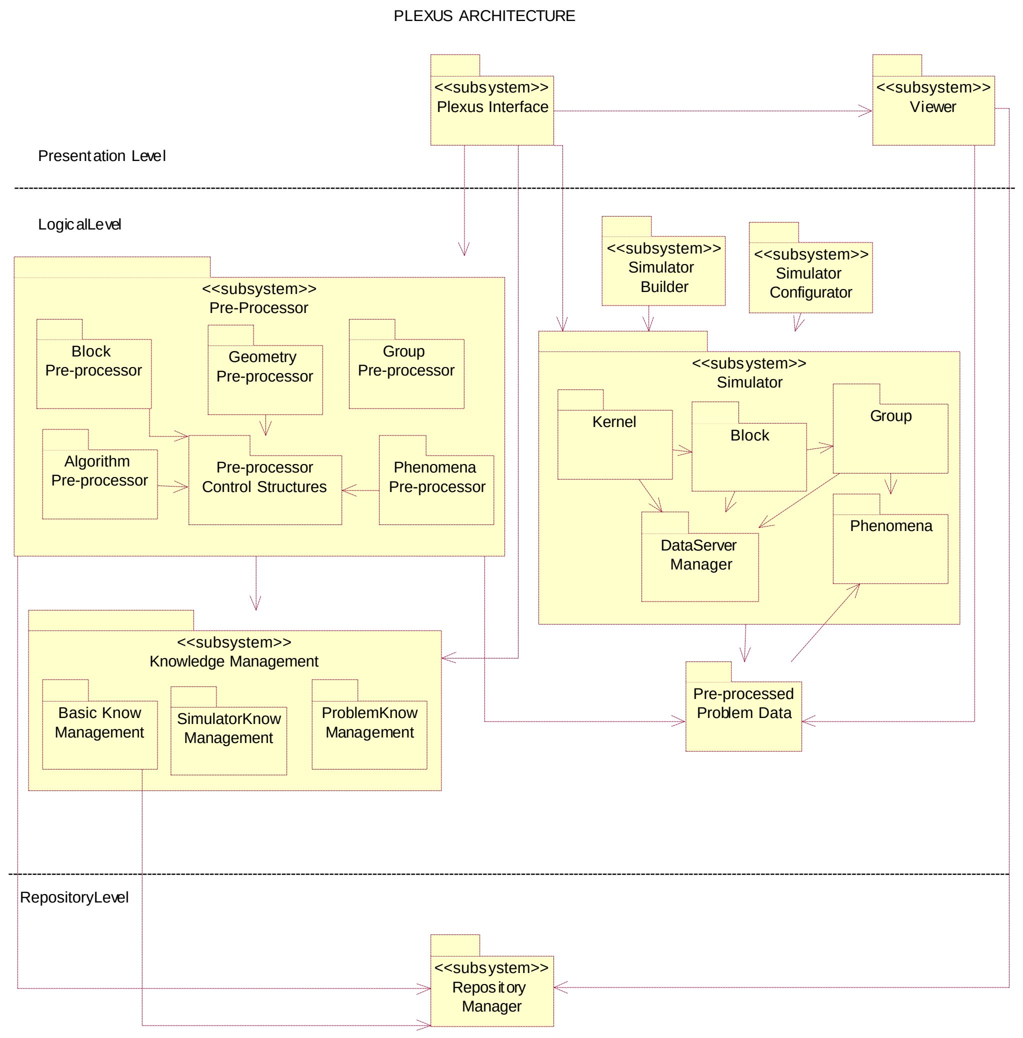


Figure 4-1 Plexus Architecture

Each subsystem's architecture is described bellow. In fact the simulator component is the main component, which can be seen as a framework. By framework, we mean a software abstraction that is defined by a set of cooperating classes that make up a reusable and customisable design for a specific class of software [AB00]. The simulator subsystem is supported by some other subsystems, which help in its construction, configuration, and data mapping and visualization.

4.2.1 Knowledge Management Subsystem

This component is the domain registration subsystem. Its purpose is to provide input and maintenance of the generic data on simulators and simulation in the system database. This data includes: basic FEM, simulator, and problem knowledge data. All this data will provide users with basic information, guiding problem data definition and simulation construction. Through the selection of data, that were previously modelled and recorded in the system, Plexus guarantees less data replication. Independence between data and programming code could be achieved. This subsystem includes:

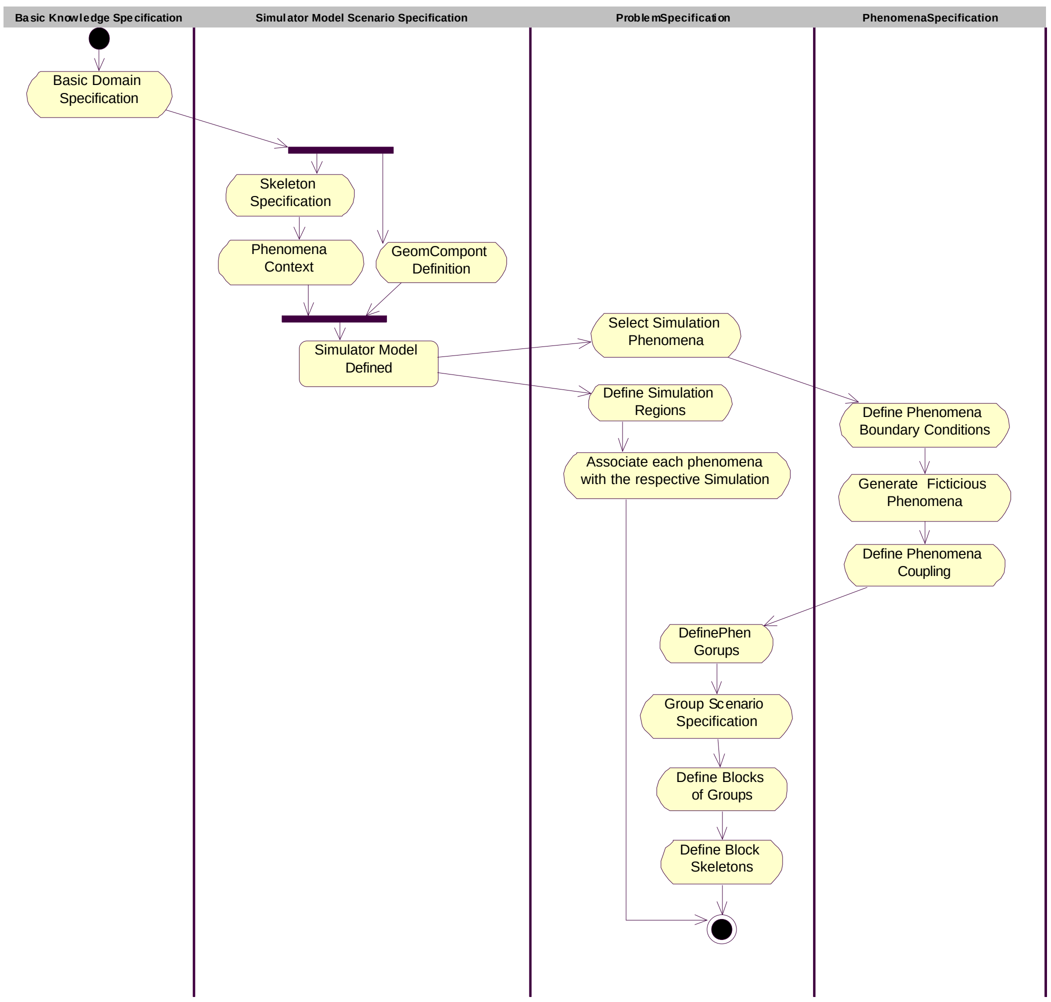
- Loading/Maintenance of basic data to the system database. The basic data acquired represents general reusable data. It defines for example different simulators meta-models, problem scenarios, and generic representations of phenomena with existing possibilities for behaviour laws. It also defines and provides reuse for components related to simulator skeletons, and numerical algorithms, etc.
- Manipulation of Strategies Catalogue: Finite Element Strategies Catalogue searches for strategies that have been previously implemented, allowing solution reuse.
- Maintenance of Simulation Problems: simulation phenomena occurring in a supplied geometry, each phenomena describing its numerical solutions, existing blocks and groups of phenomena.
- System Management: environment configuration, definition of system users, groups of users and their privileges, and importation/exportation of data, etc.

Next, we present the main sequence and dependence among input data and intermediary states achieved. The Simulator data input is composed of two main steps: Simulation Scenario Specification and Problem Specification. The description of the involved data was presented in chapter 3, in the Knowledge Base Domain section 3.4.5.

Figure 4-2 details the involved activities within 4 parts: the Basic Domain Specification, basic to all simulators, the Simulator Model Scenario Specification, the Problem Specification and the Phenomena Specification. Note that the Basic Domain Specification is related to all simulators and the Phenomena specification is a step of the Problem Specification. So we will consider only two steps, as follows:

(1) Simulation Model Scenario Specification, which in turn is composed of other specifications, which must follow a pre-defined sequence:

- Skeleton Specification where the main features of the simulation strategy are determined: phenomena classes (transient, steady, etc.), estimation error (in space, time and model), adaptation (in space, time and model), etc.
- Definition of Phenomena Context: for instance, heat transfer in solids or liquids; flow of Newtonian fluids; linear or non-linear elasticity;
- Definition of Geometry components where afterwards (in the problem specification) the phenomena will be defined. This step is independent of the two previous items.



Basic Knowledge Specification
Simulator Model Scenario Specification
ProblemSpecification
PhenomenaSpecification
Basic Domain Specification
Skeleton Specification
Phenomena Context
GeomCompont Definition
Simulator Model Defined
Select Simulation Phenomena
Define Simulation Regions
Associate each phenomena with the respective Simulation
Define Phenomena Boundary Conditions
Generate Ficticious Phenomena
Define Phenomena Coupling
DefinePhen Gorups
Group Scenario Specification
Define Blocks of Groups
Define Block Skeletons

Figure 4-2 Pre-processing Input

(2) Problem Specification (Raw data). In this step, there are three important intermediary states of defined data, related to problem specification, which can be achieved. They represent the states when: the Phenomena, Groups, and Blocks are defined. Figure 4-2 shows that the process must follow a specific sequence and that some actions can be carried out in parallel. It includes the identification of simulation regions and specific phenomena that will take part in the simulation problem, association of each simulation region with the correspondent phenomena, specification of each phenomena (boundary conditions, “fictitious” phenomena[1] and coupling) and also phenomena groups and blocks. For each group of phenomena the user must specify the scenario (front tracking, type of linear solver, equation type, etc.) that will generate specific algorithm skeletons and will constrain the group methods. There is

[1] Detailed in the Computational Phenomena, described in Chapter 5.

also the specification of Blocks, and each block has a set of skeletons, which satisfies the demands from the Global Skeleton by decoding them into demands for the groups in a previously defined order.

Figure 4-2, starts considering the definition of the basic system domain specification. In this diagram, the process reaches a final state that assumes that a defined task has been completed. An example can be seen in Appendix A (Example 1).

4.2.2 Simulator Builder Subsystem

Simulator Builder is a subsystem, which builds simulators based on algorithm skeletons, phenomena contexts, and a global simulator algorithm. After the designer has selected the simulator meta-model to be built, from the repository manager (which contains different kinds of pre-defined simulators), the builder component recovers the associated data, creates the required infrastructure for organizing simulator objects, instantiates the simulator objects, identifies involved components, and other required objects. It also makes a simulator configuration, if required by the designer at this stage.

The Simulator Builder constructs the simulator taking into account the simulator model, which distinguishes: types of solution (linear or non linear), types of Phenomena (such us transients, pseudo-transients, and permanent), requirement for Adaptation (time, space, none), inclusion of Estimation Error or not; and requirement for Mesh Generation or not.

4.2.3 Simulator Configurator Subsystem

The Simulator Configurator supports the configuration of the simulator. It allows the redefinition of the simulator configuration from designer supplied data. The designer can change the simulator articulations strategies (that is global skeletons, block skeletons and group skeletons) by selecting different ones and can change the relationship between blocks, groups and phenomena (for example change the number of blocks, etc.). However, it does not allow for modifications of the phenomena context and global skeleton specification. It includes a Simulator Set-up, which assembles the appropriate structures (graphs) that will manage the simulator process flow.

4.2.4 Pre-processor Subsystem

The Pre-processor is a tool that assembles some standardised control structures to support the mapping services. It takes input data (raw data pieces) and coverts it to the simulator pre-processed problem data (see Figure 4-3). Once the pre-processor finishes, the Pre-processed problem data is supplied to the Simulator. This, in turn, uses a service called Load Simulator data (from Simulator), which builds specific simulator data structures, implementing the format required and recognized by the simulator.

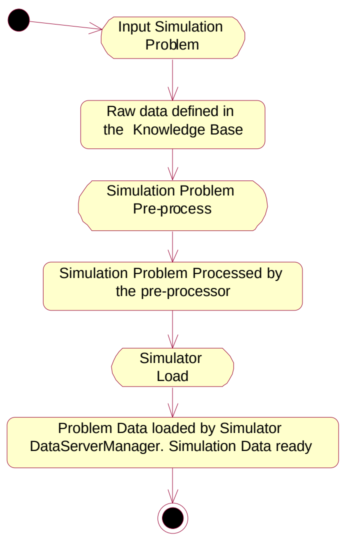

Input Simulation Problem
Raw data defined in the Knowledge Base
Simulation Problem Pre-process
Simulation Problem Processed by the pre-processor
Simulator Load
Problem Data loaded by Simulator DataServerManager. Simulation Data ready

Figure 4-3 Data transformation

Figure 4-1 shows the Pre-processor package decomposed into: (i) Control Structures package, which is related to the special structures that help the pre-processing of the raw pieces of data and transform it into suitable pre-processed problem data to be further used in simulator loading. (ii) Geometry, Group, Block, Algorithm, and Phenomena Packages. These packages are abstractions for dealing with the phenomena that occur in a given simulation region from a supplied geometry [LSA02b].

The main part of pre-processing corresponds to processes related to *dynamic structures building*, which is composed of:

- **Object Manager Pre-processing**: where some important singletons are created, such as: PhenDomainManager, GeomDomainManager, Algorithm Manager, GraphManager, PhenGroupManager, Simulation Manager, Post-processor Manager, DynamicStructuresManager;
- **Geometry Pre-processing**: creation of dynamic structures that will represent the initial geometric components of the simulation scenario;
- **Phenomena Pre-processing**: creation of phenomena managers in each simulation region;
- **Geometry Collapse**: identification of the phenomena that share a geometric mesh and collapse the correspondent geometry before mesh generation takes place;

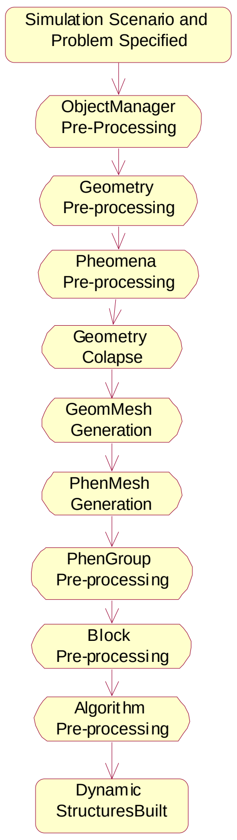


Figure 4-4 Dynamic Structure Building

- **GeomMesh Generation**: process responsible for the discretization of the geometric entities creating geometric meshes. Each phenomenon occurring in a simulation region has an associated mesh, which can be shared or not with other phenomena;
- **PhenMesh Generation,** process where the related phenomena mesh is created, that is, where the vector field approximation is defined on each geometric finite element;
- **PhenGroup Pre-processing**: creation of dynamic structures that represent each of the defined phenomena groups, each solution method and the existent priorities of execution;
- **Block Pre-processing**: definition of the groups, which will compose each block, definition of all block skeletons;
- **Algorithm Pre-processing**: the algorithm skeleton of the main process of the simulation being defined is assembled in the dynamic memory, including the solution skeleton of each block and group. The features identified in the Pre-processing Input determine the Global Algorithm Skeleton.

The final state (Dynamic Structures Built) represents that the dynamic structures have been built, and the system is ready for simulation start, since the control structures have already been assembled.

Different managers compose the Pre-processor Control Structures Package present in Figure 4-1. In the Pre-processor Control Structures package, see Figure 4-5, each manager has an exact specific function:

- The *PhenDomain Manager*, which deals with all structures that are responsible for the phenomena data manipulation;
- The *GeomDomain Manager*, which deals with all geometric data from the supplied geometry and meshes up to the slave geometry (the implementation of the simulation region where a phenomena occurs);
- The *Graph Manager*, which is a generic package that supports the system with structures and tools to deal with graph structures that are to be used in the representation of phenomena and geometry.
- The *ObjectManager*, which is a generic package that helps building complex objects used during the process.

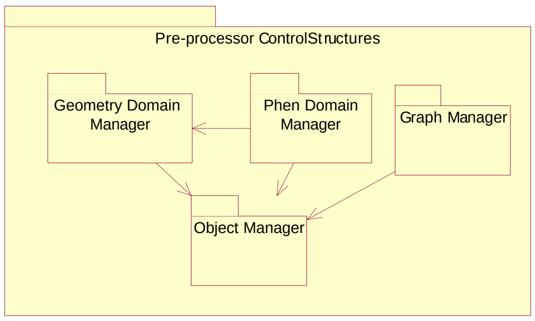


Figure 4-5 Pre-Processor Control Structures

The most important class packages, Geometry Domain Manager and Phen Domain Manager, are described in the sequence. Due to simplification constraints only some classes are explained.

Geometry Domain Manager Class Package

- Master Geometry Domain – supplied geometry, it starts hierarchically from the definition of the highest dimension geometric part (for instance, volume) down to the definition of the lowest dimension parts (i.e., points);
- Geometric Entity – it is each one of the geometric components into which is decomposed a Master Geometry Domain;
- Simulation Region– it is the highest dimension geometric entity, part of the supplied geometry, where a specific phenomenon is to be defined;
- Slave Geometric Domain: implements the simulation region;
- Slave Geometric Entity – it is each one of the geometric entities into which is decomposed a Slave Geometric Domain;

- Geometric Mesh: geometric entity that defines a discreet geometric approximation;
- Geometry Finites Elements: tetrahedron, triangles, etc;
- Geometric Domain Manager: the manager of all geometries involved.

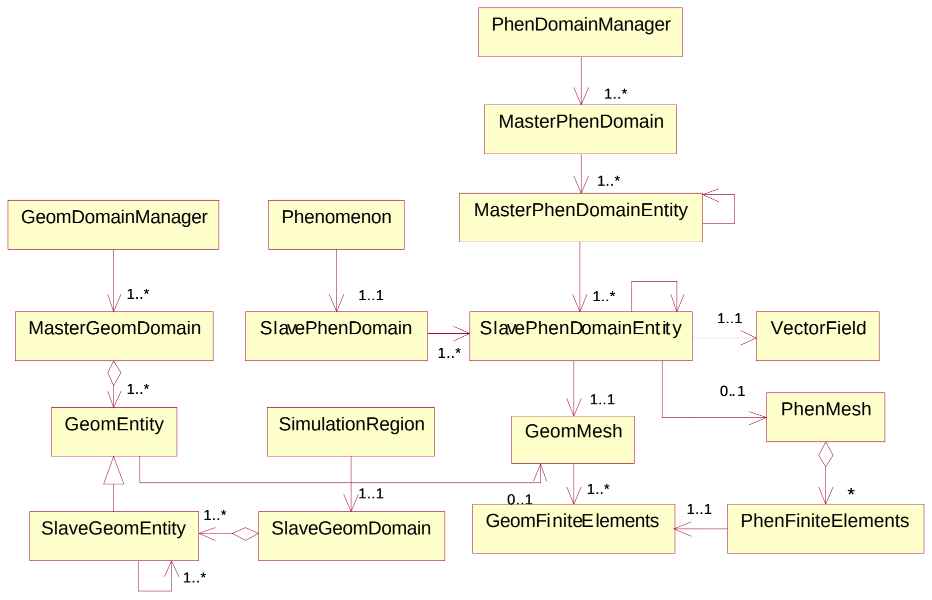


Figure 4-6 Geometry and Phenomena Manager

Phendomain Manager Class Package

- Master Phenomenon Domain is the manager of all the phenomena, which occur in a simulation region. It constructs the Slaves Phenomenon Domain and supplies each one with: a copy of the respective geometric entity; construct it's Vector Fields and Weak Forms, etc.
- Slave Phenomenon Domain– entities, which contain all the information about a phenomenon, defined in a particular Geometric Entity of the problem.
- Master Phenomenon Domain Manager: it is the manager of all Master Phenomenon Domains involved in the simulation process.

Figure 4-6 presents a view of the main structures of the Geometry and Phenomenon Packages [LSA02b].

4.2.5 Simulator Subsystem

The Simulator subsystem is responsible for the simulation of coupled phenomena applying FEM. In fact, the Simulator is a tool, which manages simulator programs process flow. Figure 4-7, which is based on [WMC95], shows that: the Simulator Kernel represents the Workflow Engine; the simulator Global Skeleton instance represents the defined workflow process; and that the Workflow Invoked Application are instances of the Block and Group skeletons, and also Numerical Methods that take part of Phenomena Data. Plexus does not consider the Administration and Monitoring tools yet. The workflow users are the designers, which define the workflow process when they build a simulator and configure it. Users are those who run simulations through the definition of the invoked applications.

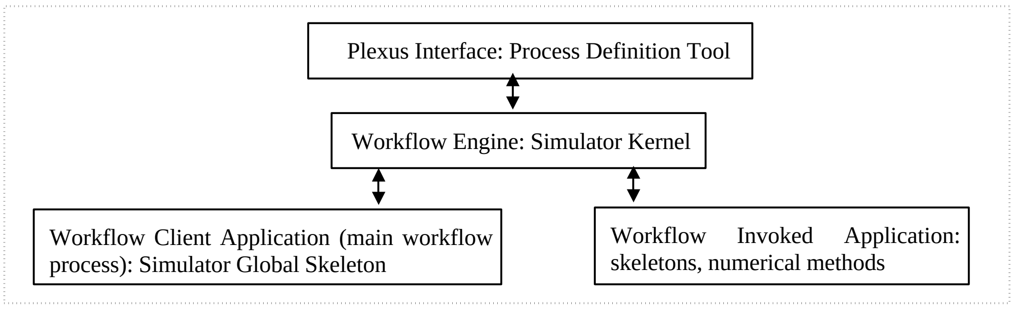

Plexus Interface: Process Definition Tool
Workflow Engine: Simulator Kernel
Workflow Client Application (main workflow process): Simulator Global Skeleton
Workflow Invoked Application: skeletons, numerical methods

Figure 4-7 Overview of Plexus Workflow Perspective following Workflow reference model

The simulator process workflow is defined, during the simulator building, through the extension of the simulator by a Global Algorithm Skeleton (which include specific processes for solving blocks and groups of phenomena) and other relevant sub-processes. The simulator functionality is supported by its algorithm skeletons and in the extension provided by the computational phenomena, which are incorporated as simulator data, and encapsulate phenomenon solution methods. The simulator applies the FEM-Skeleton and the GIG patterns, described in chapter 5. Therefore, we can represent the simulator system through a class diagram, presented in Figure 4-8, which is composed of:

- The Kernel, which will execute the defined simulator workflow process;
- The ServerManager, which supplies services for loading the simulator data in specific structures; its services can be supplied at run time or not, providing simulator adaptability for processes and data.
- Phenomena, which defines the specific and supplied phenomena strategies (numerical methods) and data.
- The Global Skeleton, which defines the workflow process.
- Blocks and groups, which constitute the method of solving phenomena together.

The solution of coupled phenomena frequently involves the decomposition of differential operators: the solution of a set of phenomena at different times, with transference of information between the solution instances. A problem that arises with the use of this technique involves the initial instant, when there is no information available about the previous solution. Then, initial solutions must be supplied. These procedures were encapsulated in the phenomena domains, and were described in this work as Computational Phenomenon patterns (which take part in the simulator as supplied data, treated by the Server Manager, see Figure 4-8).

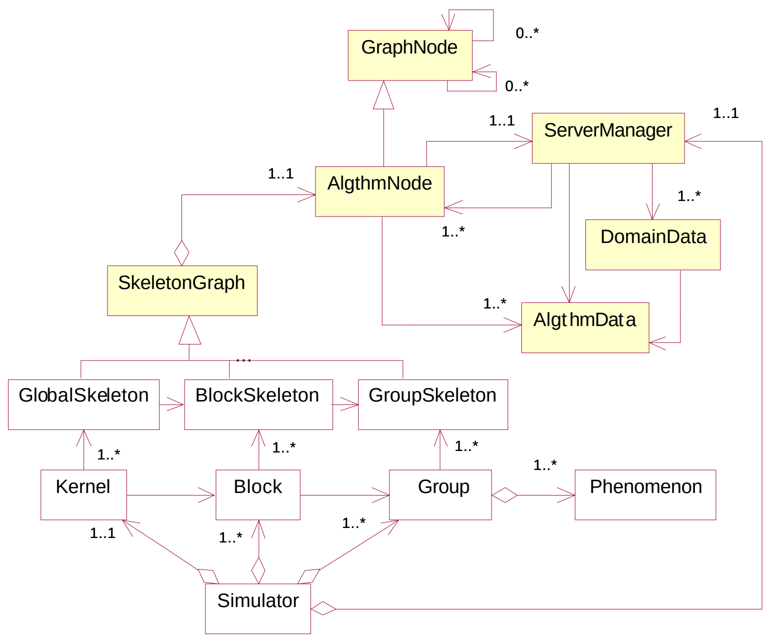


Figure 4-8 Simulator Component

Generally the main effort in the simulation run would be the computation of the discrete vector fields for all phenomena. This means the solution of several coupled systems of algebraic equations for each time step (if time dependent). The computation of extra quantities, which may be derived from those discrete vector fields, and visualization procedures may also impose considerable load on the computer system.

The simulator solution process is hierarchical. Firstly, it considers the global solution and the subsets of phenomena to be solved simultaneously (blocks and groups). Secondly, each of these phenomena is detailed. The objects needed for each one of these hierarchical stages are encapsulated in objects of the same class, called here the solution domain. In this way each phenomena has a solution domain that will manage all the processes related to the contribution of it to the regional and global solution (matrix, vectors, estimation error, and adaptation of the discrete model, etc.). The same occurs to the set of phenomena that are to be considered simultaneously (blocks and groups of phenomena), which are defined by the decomposition of the differential operators and global procedures.

For instance, the global Skeleton articulates the time loop (if present), adaptation iterations and defines processes involving the call of Block Skeletons. Block Skeletons may define different solution strategies for different Groups, thus, articulating Group processes. Group Skeletons articulate their phenomena procedures in very specific less reusable ways. It is at this level that solvers for algebraic systems are applied. Phenomena are the abstraction of the entities being simulated. All those skeletons can be implemented as objects from classes following the GIG pattern (Chapter 5). Therefore, the GIG would allow for the realization of the interoperability of the different levels of computation (by automatically plugging the lower level skeletons into the higher ones).

4.2.6 Viewer Subsystem

Viewer is a part of the Presentation Level that allows the visualization of simulation results. The solution is processed in order to allow the access and visualization of specific values, relevant to the user. These subsystem functionalities include: (1) View of data using a specific graphic tool; (2) Support for queries, considering different user needs; (3) Validation, which determines the appropriateness of the scientific principles and mathematical models used to develop simulation tools. This pattern is not part of the current work.

4.2.7 Pre-processed Problem Data

The Pre-processed Problem Data Package is responsible by the problem input data after being treated by the Pre-processor. It will be further transformed by the Simulator Server Manager to represent the simulator data (AlgthmData, figure 4-8), presented in Figure 4-3.

4.2.8 Plexus Interface Subsystem

The Interface Package is composed of all the system windows that allow access to the Plexus environment. These interface windows are presented in appendix C.

4.2.9 Repository Manager Subsystem

This Package implements functions to support the whole system with a higher level of abstraction to access data methods for the system repository.

The database maintains the general data related to meta-simulators models, the context, the algorithms that take part in different simulation strategies, the simulation problem’s data and also the simulation’s intermediary data and results. To improve process reuse, Plexus stores in the database and classifies existing processes in the following way:

- Numerical Methods, which are already known numerical algorithms that can be incorporated or implemented by the simulator. They represent algorithms available in the literature. They can be classified according to their relative context (such as geometry, phenomena, etc.). They are used, for example, to implement phenomena solution methods.

- Simple Processes, which are implemented by the designer to provide the extra functionality required, and do not need to be specific numerical method algorithms.
- Plexus Algorithms, which can be incorporated or defined through the system. They represent the composition of defined processes. They can include alternatives for a specific functionality; however there must already be defined default processes.
- Skeletons, processes defined by the designer, which will compose the main structure of the simulator. They can be a specialization of the above-defined Plexus Algorithm. In pursuit of a high degree of reusability, hierarchical levels of processes are used to define a simulator, where each level may have several possibilities of algorithms, and can be easily described by a graph. These levels satisfy a number of requirements, such as: (i) to separate less reusable modules from reusable ones; (ii) to help the understanding of the decomposition of the simulation data among the several processes; (iii) to make possible the dynamic re-configuration of the simulator through the replacement of reusable modules. More details can be seen in the FEM-Simulator Skeleton (chapter 5).
- Built and configured algorithms, which correspond to a designer strategy mounted from the composition of existing numerical methods, components, or simple process, but which do not contain alternatives (the process was already defined), in this way they are different from simple Plexus Algorithms.

4.3 Plexus Architectural Behaviour

As described in previous chapters, the Plexus main processes consist of Knowledge Management, Simulator Management, Pre-processor, Simulation, and Simulation Results View.

Figure 4-9 presents part of the collaboration diagrams, which describe the main processes, their inter-relationship and sequence of subsystem execution, which are detailed in the following items:

1. The Plexus controller starts requesting the Knowledge Base Manager services for storing data (basic data loading, simulator specification and problem data definition).
2. The Plexus Controller requests the simulator construction for the Simulator Builder.
3. The Simulator Builder requests the simulator related data to the Repository Manager. The retrieved data is relative to the details of the selected simulator.

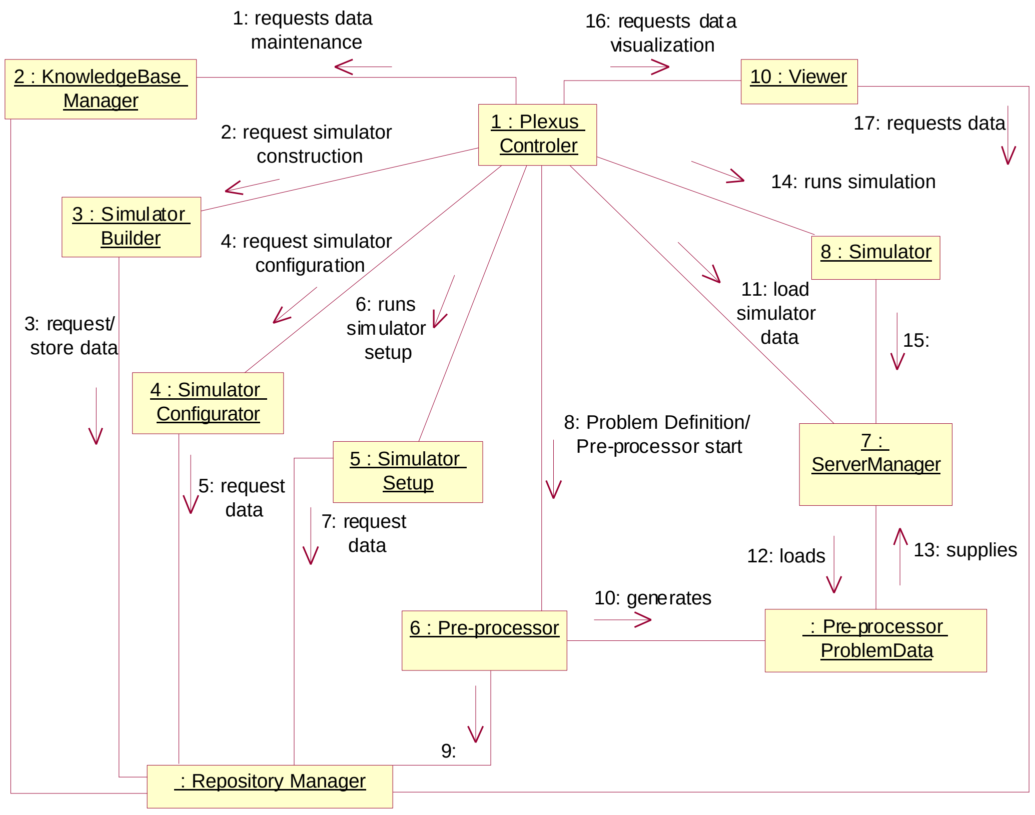


Figure 4-9 Plexus Functioning

4. The designer wants to specify the articulation strategies, the Plexus Controller requests the system configuration (however, this can be done later).
5. The Simulator Configurator acquires the required data from the Repository Manager.
6. Then, the Simulator Set-up is executed. This will guarantee that the basic simulator structures are created.
7. Data is requested to the Repository Manager.
8. The ProblemDefinition/Pre-processor can be started.
9. Data is requested to the Repository Manager.
10. The Pre-processor Data is generated (see details given previously in Figure 4-3).
11. The simulator data is loaded; the Server Manager transforms the pre-processed data into the appropriate simulator structures (it is part of the simulator component, see Figure 4-8).
12. The data is supplied to the Server Manager.
13. The Simulation running is started.
14. The data is requested to the Server Manager.
15. Simulation data visualization is requested.

16. The simulators viewer can be requested to present the required visualization data.
17. The viewer in turn requests the appropriate data from the repository manager, which in turn supplies it.

The Simulator Builder and the Simulator define specific sub-processes. However, their logic is very re-configurable, that is, most of the sub-tasks are changeable.

The definition of a process based on FEM allows the control of coupled phenomena, guaranteeing a high level of abstraction and reuse of developed solutions. The complexity of the implementation phase can be greatly reduced by the use of a predefined process that describes the activities and stages that take part in the complete process of defining and constructing the main structures that are part of the Plexus control system.

The following section describes an evaluation of Plexus architecture.

4.4 Plexus Architectural Evaluation

For an architecture to be successful, its constraints must be known and it must not only implement the functional requirements of the system which are specified in the requirements document, but also satisfy many quality attributes, such as whether or not the system will have to interact with other systems (interoperability), and meet business goals (such as future desires), etc. Bellow we evaluate our proposal architecture to affair the previous identified objectives.

Table 4-1 Summary of the Architecture Evaluation

Problem	Solution
Single Software Solution for coupled multi-physic simulation based on FEM	Definition of an architecture incorporating subsystems that apply FEM-Skeleton Simulator Pattern; Computational Phenomenon Pattern, and the Pre-processor Pattern.
Flexibility in the implementation of different numerical methods and definition of different simulators strategies	Implementation of simulator modularity, through different levels of computation, which allow the customisation of numerical solutions (FEM-Skeleton Simulator). This allows the configuration of the: (i) simulator articulation strategies configuration (Configurator Component) (ii) phenomena numerical methods (Computational Phenomena Component). Support for the dynamic adaptation of the system workflow.
Make the simulators and problem development easier and faster than before	Pattern definitions for the FEM simulators domain development providing reusable solutions; Improvement of Domain understanding and definition; Reusability of numerical solutions, models and data, supported by a database repository; Simplification of Requirements; Focus on application (specific architecture considering

	defined patterns). Automatic Routines (not considered in this work).
Distribution	Distribution of solutions (partially treated by defined Architecture); (ii) Encapsulation of activities in specific abstractions; Simulator modelling and implementation through computation levels make the distribution of functionalities and the simulation process easier.
Persistence	It is considered through meta-models definition and its implementation is supported by the use of a Data Base Management System.
Integration with other systems	The system allows the incorporation of new components and due to its modular organization allows easy extension to new functionalities.
Maintainability	Definition of the architecture in levels (hierarchical) and in subsystems helps system maintainability.

Plexus architecture considers functional and non-functional requirements, which drive its definition. The functionalities are related to the implementation of all the processes required to develop a simulator based on FEM. However, the conceptualisation of the architecture articulates many non-functional requirements to achieve its purpose. Table 4-1 summarizes some of them giving an overview of the adopted solutions.

4.5 Final Considerations

In order to achieve an effective development process in software engineering, reuse and abstraction can be applied at many different levels of design. The software architecture level is specifically concerned with the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints of these patterns. Plexus architecture is a domain specific architecture for FEM simulators. It applies the object-oriented paradigm, for constructing families of applications, in our case simulators.

There is a well-known specification architecture for composite simulations [KWD99] called High Level Architecture (HLA). In pursuit of simplification, it is not the purpose of this work to include HLA integration. We only take into account its features, considering it as a future improvement for the Plexus architecture. The purpose of the HLA is to support requirements related to the increasing need to build *more complex* and *realistic numerical simulations* (such as interoperability and distribution). In this architecture, components are individual simulations. Each one can be developed and executed independently, and when integrated they build a major simulation. It supports the building of simulations distributed across multiple computers. However, nothing in the architecture assumes or requires a distributed implementation of a simulation. HLA development was based on an initiative involving government, academia, and industry. In 1998, the Defence Modelling and Simulation Office adopted version 1.3 of the HLA specification. These specifications formed the basis for draft IEE standards for simulation interoperability architecture. A more detailed description can be seen in [KWD99].

The next chapter details some of the Plexus architectural abstractions (patterns and frameworks). The applied patterns include: *modelling patterns*, for addressing framework domain features, by supporting the expression of an abstract domain model underlying the framework and *design patterns*, for addressing structural properties of frameworks by supporting the development of the logical structure [FAY99].

Chapter

5

Plexus Simulation Environment Abstractions

This chapter presents some basic concepts about Plexus architectural abstractions including patterns and frameworks. Some patterns were indentified during the Plexus Simulation Environment conceptualisation: Computational Phenomenon, FEM-Simulator Skeleton, and GIG-Patterns. The first and the second one are domain specific patterns, which help control the simulation of coupled phenomena based on FEM solutions. The GIG pattern is not domain specific.

5.1 Introduction

Software architectural abstractions can be exemplified and divided into various categories according to a set of dimensions, which include level of abstraction, degree of domain specificity, level of granularity, and the degree of completeness. Design patterns [GHJ95] and frameworks [JF88] are examples of those categories, which are applied in Plexus conceptualisation.

Patterns for software development are one of the *hottest topics* that emerged from the object-oriented community. According to [AB00]: “Patterns are literally a form of software engineering problem-solving discipline that has its roots in a design movement of the same name in contemporary architecture, and the documentation of best practices and lessons learned in all vocations”.

Being able to analyse and build a system with a regular set of building blocks, which represent common pattern usage, provides substantial benefits. These include better human comprehension of complex systems, with the reduction of the cognitive burden [BCK98], and aids both development and maintenance of the system. Such a regular set of building blocks represents common pattern usage.

A pattern is a small collection of atomic units and a description of their relationships. In general, a pattern is a named perspective on a subject. In order to be relevant, a pattern must express a general recurrent theme that has proven to be useful. Focusing on the analysis, design, and implementation efforts in software development, the subject is a problem domain, a system design, or a program implementation. In addition, the subject can be a structural diagram with classes, objects, and their relationships. Other types of subjects could be models of problem domains, interaction diagrams, or source code [FAY99].

Frameworks are closely related to design patterns. Frameworks are an object-oriented reusable analysis or design of all or part of a system, which has been used successfully and which forms an important part of the culture of experienced object-oriented developers [FAY99].

Modelling patterns address framework domain features by supporting the expression of the abstract domain model underlying the framework. A modelling pattern names a reusable abstraction over classes, specifies the static structure of the abstraction in terms of methods and class relations, exemplifies the abstraction, and provides hints on applicability [COD92, FAY99].

Design patterns address structural properties of frameworks by supporting the development of the logical structure [FAY99]. Many design patterns are related to abstract coupling or the management of recursive structures, which are issues considered in the more technical phases of design and in the implementation of frameworks. The primary focus when using design patterns is the development of frameworks. Design patterns promote loose coupling between the parts, making framework design flexible. To provide flexibility in terms of abstract coupling between the constituent parts, a design pattern is used to design the functional factoring and component interfaces in a part of the framework. The design patterns address very explicitly the task of providing a framework with structural characteristics.

A collection of patterns, built on each other to generate a system, form a pattern language [ALX77, COP01]. A pattern in isolation solves an isolated design problem; a pattern language builds a system. It is through pattern languages that patterns achieve their fullest power. A pattern language describes an architecture, a design, a framework, or another structure. It has structure, but not the same level of formal structure that one finds in programming languages. A pattern language is not just a decision tree of patterns. This is partly because the patterns of a pattern language form a directed acyclic graph (DAG), not a hierarchy. The number of distinct paths through a pattern language is very large. It is indeed, the structure of this network, which makes sense of individual patterns, because it anchors them, and helps to make them complete.

We can identify a collection of patterns that generate the Plexus system environment. The Plexus Pattern Language can be decomposed into the following patterns (see Figure 5-1): (i) FEM-Pre-processor Pattern; (ii) FEM Simulator pattern, composed of FEM-Simulator Skeleton and the GIG-Patterns; (iii) Computational Phenomenon Pattern; (iv) Viewer Pattern.

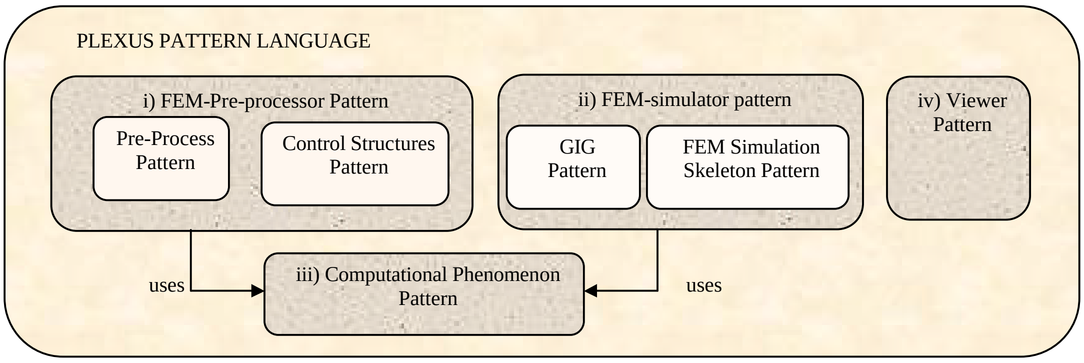

PLEXUS PATTERN LANGUAGE
i) FEM-Pre-processor Pattern
Pre-Process Pattern
Control Structures Pattern
ii) FEM-simulator pattern
GIG Pattern
FEM Simulation Skeleton Pattern
iv) Viewer Pattern
uses
iii) Computational Phenomenon Pattern
uses

Figure 5-1 Plexus Pattern Language

The Computational Phenomenon Pattern, the GIG-Pattern, and the FEM-Skeleton Pattern are detailed in the rest of this chapter. These patterns deal with specific problem solutions such as Simulator Modelling and Workflow definition and control.

- The Computational Phenomenon Pattern represents the phenomenon model to be used by the simulator. The main objectives of such an abstraction are to have a common language to represent a phenomenon in the coupled context and to make intuitive and easy the representation of data sharing and dependence between different phenomena.
- The objective of the FEM-Simulator Skeleton Pattern is to guide the development of Simulator Models based on FEM. This pattern is intended to help the design and implementation of simulators. The main advantage of the pattern is its high level of abstraction, reusability, and modularity in the design of simulators for several coupled multi-physics phenomena.
- The GIG Pattern represents a solution based on a generic interface graph, which deals with the definition and control of processes flow, taking into account some specific requirements of simplicity, making easier the definition from algorithmic natural language and giving flexibility in the granularity of defined processes. The pattern is intended to help the design and reuse of programs. It is not domain specific.

The other patterns, that is the FEM Pre-processor Pattern (described in [LSA02b]) and the Viewer Pattern (described in [VA02]) are still under development and are not presented in this work.

There is a lot of confusion about whether frameworks are just large-scale patterns, or whether they are just another kind of structure. A frequently used definition is "a framework is the skeleton of an application that can be customized by an application developer". The term "framework adaptation" is applied when a developer customizes a framework to a particular application [FAY99]. Both definitions for frameworks are not conflicting; the first describes the structure of a framework while the second describes its purpose [FAY99]. When structuring a framework in terms of design patterns, the framework structure will be made visible because patterns will describe logical units and point out abstract couplings that take part of the framework Figure 5-2. This is important for both framework adaptation and framework evolution.

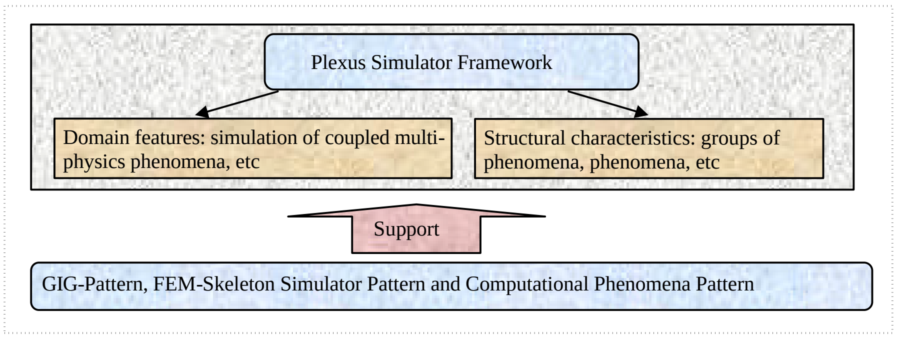

Plexus Simulator Framework
Domain features: simulation of coupled multi-physics phenomena, etc
Structural characteristics: groups of phenomena, phenomena, etc
Support
GIG-Pattern, FEM-Skeleton Simulator Pattern and Computational Phenomena Pattern

Figure 5-2 Frameworks supported with patterns

A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations [GHJ95]. A specialist in a particular framework sees the world in terms of the framework and will naturally divide it into the same components [FAY99]. The adaptation of a framework is done through the customisation and extension of the framework structure. The parts of the framework that are open to extension and customisation are termed *flexible hot spots*. They express aspects of the framework domain that cannot be fully anticipated. They are discovered during domain analysis or provided by a domain expert. The components to be supplied in the flexible hot spots can be components from a library belonging to the framework (providing different alternatives), and/or they can be user created. A framework can be adapted by means of white box or black box reuse. The former corresponds to customizing the framework classes by specialization, and the latter corresponds to configuring a part of the framework.

Plexus environment applies the object-oriented paradigm for constructing families of FEM simulator applications, which can benefit from the framework concept through the exploration of system customisation. In addition, it provides the support for the understanding and construction of simulator systems, which include the database models, the system interface (see appendix C), and the involved processes classification (for example skeletons and numerical methods described in chapter 4, section 4.2).

The Plexus architecture includes the simulator subsystem component, which can be seen in Figure 5-3 as a framework abstraction. This framework, applies the GIG and FEM Simulator Skeleton, and the Computational Phenomena Patterns (which can be seen in Plexus Pattern Language, see Figure 5-2). In fact, the Simulator Framework includes a component, the Computational Phenomena Framework (see Figure 5-4), which applies the Computational Phenomena Pattern.

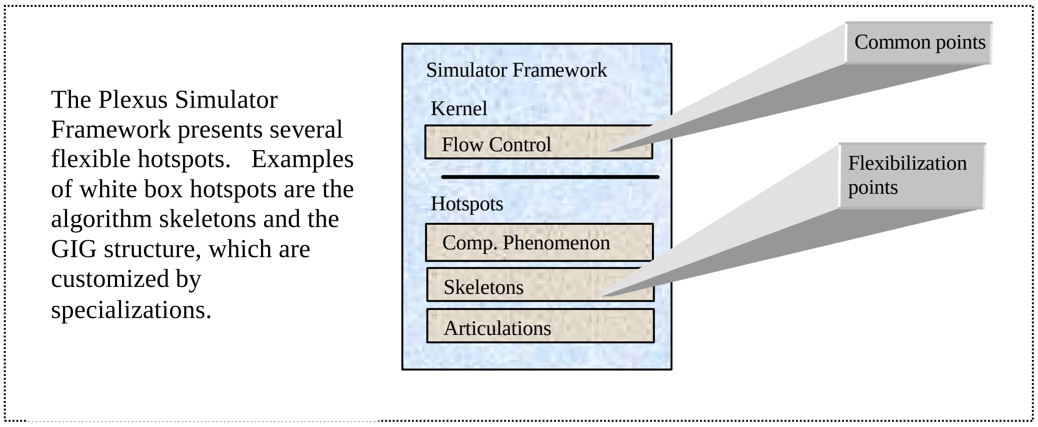

The Plexus Simulator Framework presents several flexible hotspots. Examples of white box hotspots are the algorithm skeletons and the GIG structure, which are customized by specializations.
Simulator Framework
Kernel
Flow Control
Hotspots
Comp. Phenomenon
Skeletons
Articulations
Common points
Flexibilization points

Figure 5-3 Plexus Simulator Framework

Figure 5-3 and Figure 5-4 give an overview of both Frameworks identifying the existing hotspots.

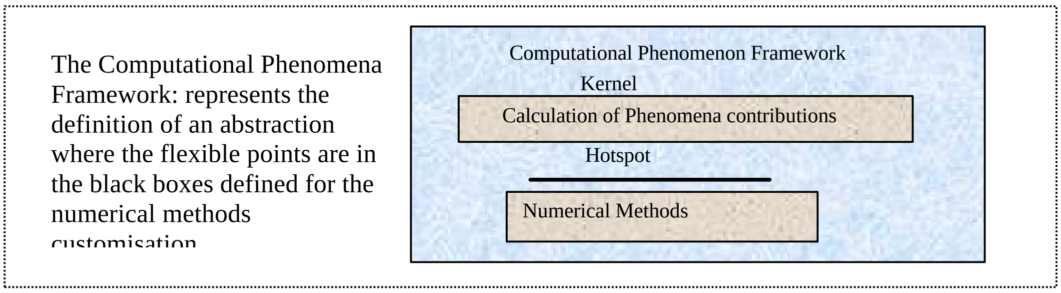

The Computational Phenomena Framework: represents the definition of an abstraction where the flexible points are in the black boxes defined for the numerical methods customisation
Computational Phenomenon Framework
Kernel
Calculation of Phenomena contributions
Hotspot
Numerical Methods

Figure 5-4 Plexus Computational Phenomena Framework

This chapter presents some specific patterns developed considering the domain requirements. Each pattern was described in the following form based on suggestions found in [COP01]. First, the pattern name is supplied. Then, some details are given about the context in which existent problems might inhibit further developments, and to which the pattern solution applies. After, the design challenge is presented through a question. Then, pattern forces are shown, that is, the patterns design trade-offs, which pulls the problem in different directions, towards different solutions. Next, an explanation about how to solve the problem is presented. Then, the pattern applicability is described. In the sequence, an example of usage is presented. Then, the resulting context is detailed, indicating which forces the pattern resolves and which forces remain unresolved by the pattern. Finally, related patterns are shown and some comments are made on about known users.

The rest of this chapter is organized as follows. Section 5.2 presents the Computational Phenomenon Pattern. Section 5.3 introduces the FEM-Simulator Skeleton Pattern. Section 5.4 describes the GIG Pattern. Finally, in section 5.5 we make some closing remarks.

5.2 Computational Phenomenon Pattern

This pattern represents an abstraction of the collection of commonalities found in the concepts and processes for representing phenomena simulation through FEM. A further objective with such an abstraction is to make the representation of data sharing and dependence between different phenomena intuitive and easy. This pattern complements the Plexus FEM Skeleton Pattern (detailed in the next section), which advocates the separation of the simulator process levels into different levels of programming, considering one specific level for the computational phenomena processes.

5.2.1 Pattern Name

Computational Phenomenon, which means a pattern for modelling the simulation phenomenon data and processes for simulators based on FEM.

5.2.2 Context

As described in chapter 2, in the definition of a simulation problem the user defines the involved physical **Phenomena** (e.g. fluid flows and heat transfer). The discrete formulation of a phenomenon comprises a set of algebraic equations (either linear or non linear) obtained through the application of a finite element technique to the exact mathematical formulation. The application of such techniques can be described using FEM concepts: finite elements, shape functions, discrete weak forms, discrete vector fields, nodal values, couplings, linear solvers, non-linear solvers, error estimation, adaptivity, and time progression schemes.

In FEM simulation, the solution algorithms can and should be defined in a modular structure and in such a way that avoid couplings between procedures at different levels:

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

The processes at the lowest level, that is, the finite element level, motivate the Computational Phenomenon pattern definition. Those processes are related to the production and assembling of the corresponding matrices and vectors. The element matrices and vectors may be coupled with other phenomena, meaning that the computations of those quantities need pieces of information from other phenomena. Since the coupling requirements can be classified and standardized, the production of uniform interfaces between coupled phenomena became possible. These interfaces can be configured (specialized) by the processes, which are going to use the Computational Phenomenon Patterns objects.
This pattern solution can be used to implement a framework that, during the pre-process phase, automatically associates any pre-defined phenomena to geometric entities in a simulation.

5.2.3 Problem

What information, relationships and processes must be supported, and in what way, in order to describe and implement computational phenomena, considering coupled multi-physics systems?

5.2.4 Forces

With respect to the definition of an adequate abstraction for computational phenomena considering coupled multi-physics systems, there are different forces, which lead to different solutions, such as:

- Possibility to replicate numerical studies.
- Achievement of higher levels of sophistication in simulation design.
- Reliability of complex computer-generated simulations.
- Reuse, extension and configuration of models.
- High levels of abstraction, modularity and the right separation of concerns.
- Automation in dealing with coupled phenomena definition.
- Simulation performance and maintainability.

The support for **numerical studies replication** can be obtained through the definition of standard solutions, persistency of knowledge and data as well as high degrees of completeness and reuse of the involved abstractions. However, those techniques may produce undesirably rigid systems. The completeness of the considered information, for a computational phenomena abstraction, can also generate a trade-off between the levels of considered concepts and rigid systems. To achieve completeness the modeller can try to gather to many details for a correct specification of the Phenomenon and its relationships within the simulation system. However, it is difficult to establish what is considered complete, that is, when it is time to stop introducing information details. Conversely, sometimes the simplification of an abstraction allows it to be used in a more extensive number and types of simulation applications and thus helps the establishment of a standardized abstraction of a model. Alternatively, completeness of abstractions supports the understanding, use and discussion of the involved details. The correctness of the abstraction provided, itself provides more reliability to simulations.

Since the conceptualisation of a system takes care of the most critical (complex and detailed) aspects of the problem under consideration, **higher levels of sophistication** can be achieved. Reuse, extension and configuration of models help in this task. However, other aspects should be considered, for instance, the control load required in order to maintain the dynamics of the system functioning.

Reliability of a simulator comes as a result of several aspects, from the system conceptualisation to the software programming and use. Techniques such as standardization of solutions, completeness of data and reusability can increase the reliability of the system.

In the context of many coupled phenomena, the set of data and procedures related to the computation of matrices and vectors (used in the solution algorithm) is the main obstacle **to reuse, extension and configuration** of phenomena models, due to the complexity of data sharing and dependence between those phenomena. **High levels of abstraction, modularity and the right separation of concerns** are important aspects in obtaining good solutions in this respect. Performance is a concern whenever those techniques are applied.

Some of the complex set of information a phenomenon deals with requires a great deal of effort in their development and are extremely reusable. The most technologically intense parts of Phenomenon can be made persistent for further use, if there is a reasonable and standardized way of representing phenomena concept models.

Also the phenomenon variants and complexity of their sets of data and procedures can make the **automation of the phenomena definition** and the building of their set very demanding in the context of coupled phenomena. The variety of possible solutions requires a careful understanding of the subjacent logic, which makes the phenomenon concept - in the finite element context - a valid concept, in the sense that it allows for a representative abstraction. Consecutively such a representative abstraction provides support for the automation of phenomena definitions.

Achieving **high performance** usually requires resolving low level machine-dependent details. Whenever methods are considered in the direction of dealing with highly complex systems requiring high levels of reusability, the performance is usually affected negatively. However, careful design procedures and considerations in the direction of distributed and parallel computations can reduce the most severe shortcomings. Performance at the level of the procedures responsible for computation of matrices and vectors in the finite element context is critical for large systems.

Maintainability for complex systems is of the utmost importance in order to make possible the tasks of error detection and correction, model extensions, system configurations etc, with a minimum amount of work.

5.2.5 Solution

A solution for the proposed problem can be achieved by the definition of a Computational Phenomena abstraction whose purpose is to reflect an adequate separation of concerns in FEM simulation modelling, preserving the encapsulation of data and processes concerning only the *numerical modelling of a phenomenon*. Data related to the numerical modelling of a phenomenon are used in different phases of the simulation. For instance, in the pre-processing phase, pieces of phenomenon data are used for the establishment of the right coupling between phenomena and for the definition of the mesh generation method. Alternatively, processes responsible for the computation (at the element level) and assembling (in the global entities) of the right vectors and matrices (coupled or not) are also specific for a phenomenon.

The separation of concerns means the separation of the data and processes, employed in the global solution of the coupled algebraic (time dependent or not) systems, from the data and processes of the numerical modelling of a phenomenon. This separation allows for the reuse of the numerical modelling data of a phenomenon in either different solution strategies or different simulations.

Subsequently, the Computational Phenomenon is considered as a set of data and processes, which define the numerical modelling of a phenomenon. The numerical modelling data of a phenomenon is comprised of a vector field, weak forms (for different vectors and matrices, shape functions, coupling data), and methods (e.g., mesh generation, approximation generation and numerical integration rule), etc.

The notation of phenomena may be abstracted (described in this work as "fictitious" phenomena) in order to allow for a generic representation of either relationships between different phenomena, or restrictions, or additional vector fields and correspondent behaviour laws (such as Lagrange multipliers). This extension of the notation of phenomena becomes very important in computational modelling, because it allows several different techniques to be used in order to impose restrictions (e.g., boundary conditions, and constitutive restrictions, etc.).

The discrete behaviour law may have terms, which are defined on the boundary of the domain. Those terms are provided by sub-phenomena, which are defined as objects, which are dependent on the correspondent main Computational Phenomena. Boundary terms are related to boundary conditions or boundary restrictions. This work considers many kinds of boundary conditions. The most well known are Dirichlet, Neumann and Mixed ones. The Dirichlet boundary conditions are considered as restrictions defined on correspondent parts of the boundary. The restrictions are modelled using a Lagrangean formulation; which requires the implementation of extra independent phenomena. The other types are modelled as sub-phenomena defined on the correspondent parts of the boundary (the fictitious phenomena is basically the same as the original physical phenomena). Even the restrictions on the boundary (like Dirichlet boundary conditions) produce coupled terms on the behaviour laws of the original phenomena. These terms are provided by sub-phenomena defined on the correspondent part of the boundary.

Participants

The Computational Phenomena pattern is composed of several participants. We divide our explanation in three parts:

- **Geometry participants** (see Figure 5-5), composed of: GraphNode, GeomEntity, GeomGraph, Point, Curve, Surface and Volume.
- **Phenomenon participants** (see Figure 5-6), composed of objects which represent phenomena, simulation regions and their interaction: PhenGraph, PhenEntity, GeomGraph, GeomEntity, PhenMethod, VectorField, Group,

WeakForm, PhenMesh, GeomMesh, GraphNode, QData, SimulationRegion, Geometry, Phenomenon, and VecFieldTransferToolsAndData.

- **Mesh participants** (see Figure 5-7), composed of objects related to the geometric and phenomena mesh: GeomFiniteElement, GeomReferenceElements, GeomIntrinsicIntegData, PhenReferenceElement, PhenIntrinsicIntegData, PhenShapeFunction, and PhenElementData.

For the details of many concepts, used in the description below, see Chapter 2 definitions.

A **phenomenon** occurs in a geometric domain. This work uses a computational representation for the geometry (geometric domain) based on the boundary representation method (Brep). This is extremely useful due to a number of reasons. For instance, when a phenomenon is defined on a region, its boundary conditions will be defined on all parts of its boundary. Thus, the relation between a phenomenon and its boundary conditions resembles that of the boundary representation (for example in a direct acyclic graph implementation). The geometry diagram is considered as a direct acyclic graph pattern (DAG). It is represented hierarchically through a graph scheme where each graph node stores a geometric entity (**GeomEntity**, i.e., point, curve, surface, and volume) see Figure 5-5. The connected components of the geometry are referred to as **GeomGraph**.

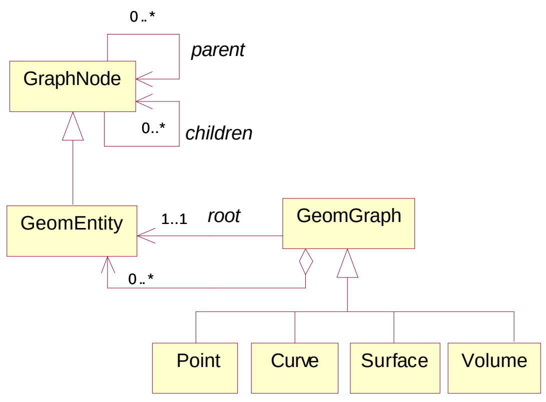


Figure 5-5 Geometry Participants (Brep graph)

A defined Phenomenon is implemented by a structure, called a **PhenGraph,** and occurs in a given Simulation Region, which is related to a Geometry (represented by a **GeomGraph**). A Phenomenon applies some specific methods for its solution, integrated in a **PhenMethod;** they include: Integration Rule, PhenMesh Generation Method, GeomMesh Generation Method, and PhenShape Function, etc.

A simulation region (**SimulationRegion**) is a partition of the geometry where a phenomenon is first defined. For each simulation region a **PhenGraph** manages the simulation data of all phenomena defined therein. A PhenGraph can also be viewed as a graph-like data structure, where each node stores a PhenEntity. The PhenGraph closely resembles the organization of the respective simulation region. Each phenomenon has a copy of its simulation region (GeomGraph). Each GeomEntity of a GeomGraph has a manager entity (the **PhenEntity**) responsible for its relationship with all pertinent simulation data. Thus, each PhenEntity has a GeomEntity where its weak forms are defined.

The **VectorField** stores basic information about the discrete vector field, such as its dimension and the dimension of its vector of nodal values. The PhenEntity needs those pieces of information for the computation of its matrices and vector sizes and for assembling them into global matrices and vectors. The **WeakForm**s represent parts of the discrete behaviour laws, boundary conditions and other pieces of information, which are needed by the solution algorithms. Therefore, the production of the phenomenon vectors and matrices (at the finite element level) and their assembly into global vectors and matrices are performed by the respective weak forms, which have knowledge of the required coupling data.

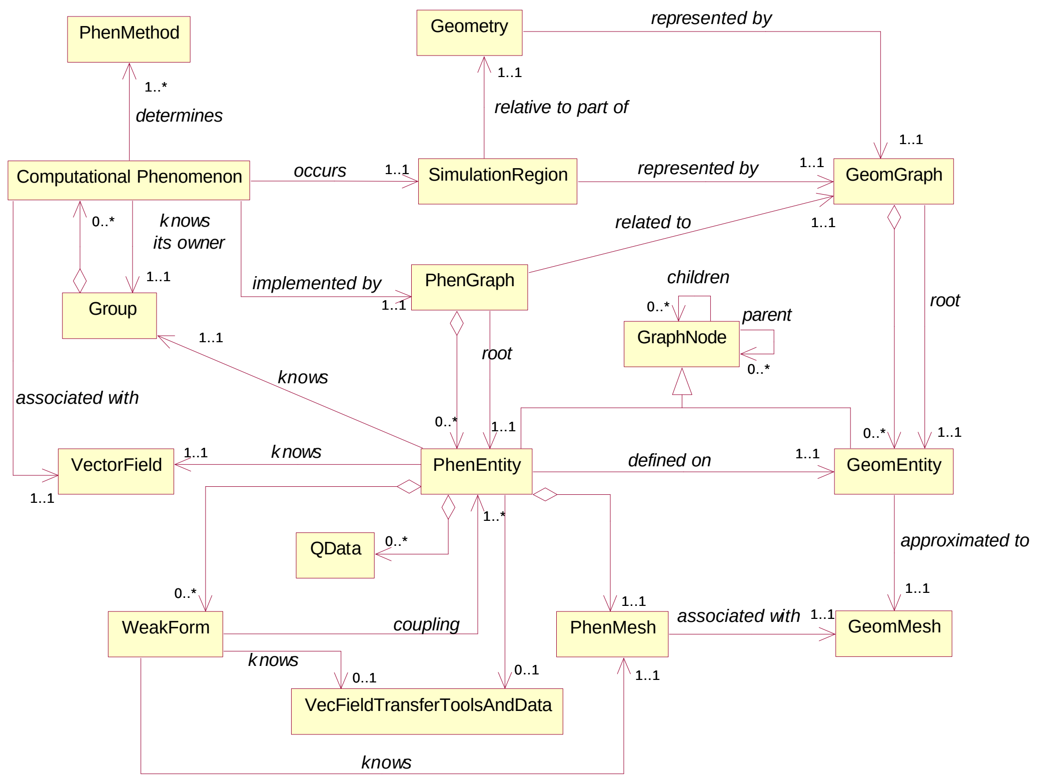


Figure 5-6 Phenomena Participants

When asked to compute and assemble a certain quantity (vector or matrix), a Computational Phenomenon sends the request to the PhenEntity in the root of its

PhenGraph. It will look in its **QData** in order to see if it is able to provide what its being asked. If so it executes the right weak form. After it is finished it passes the request to its children PhenEntities. Thus, the union of all QData objects from all PhenEntities of a phenomenon comprises the set of all quantities that the Computational Phenomenon is able to compute. QData also contains information regarding the coupled phenomena needed in its computation.

When computing a quantity, a weak form may need information from other phenomena (coupled information). Usually it will need the vector of nodal values and the phenomenon mesh from each coupled phenomenon. However, those meshes are frequently different from the current phenomenon’s mesh. Thus, in order for the incoming information to be useful, it should be transferred to a phenomenon mesh, which has the same geometric mesh as the current phenomenon’s geometric mesh. Therefore **VecFieldTransferToolsAndData** is provided with specialized tools for performing those types of tasks.

A geometric mesh (**GeomMesh**) is an approximation to a partition of a geometric domain (GeomEntity). It is described by a set of geometric finite elements (**GeomFiniteElement),** which can be see Figure 5-7.

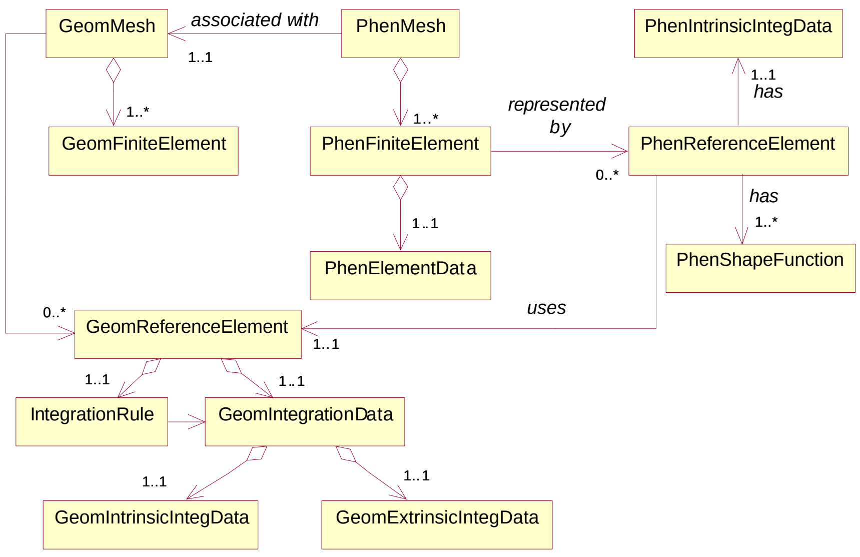


Figure 5-7 Mesh Participants

The geometric mesh depends on associated **GeomReferenceElement**, which in turn have an integration rule (numerical integration method) and numerical integration data (integration points and weights for a specific approximation order). A GeomReferenceElement has the following associated data concepts: (a) Intrinsic data (integration points and weights); (b) Extrinsic data for a given geometric finite element

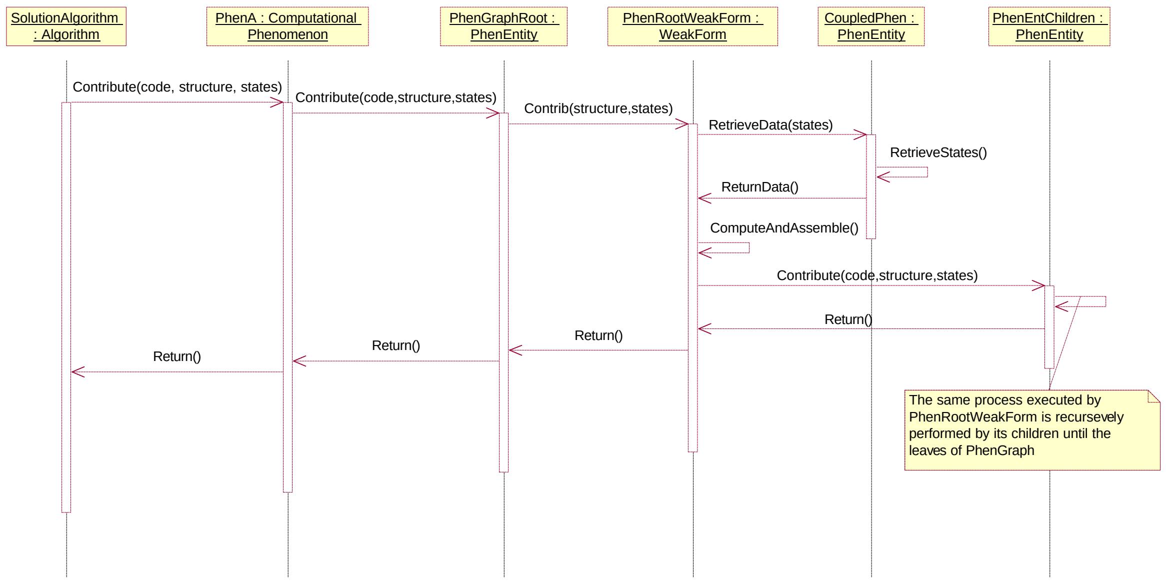
(Jacobean matrix and Jacobean at the integration points; positions of the integration points in the given finite element). Intrinsic data are fixed, while extrinsic data are dependent on the geometric finite element under consideration. The intrinsic set of integration data is encapsulated in **GeomIntrinsicIntegData**.

The phenomenon mesh is a set of phenomenon finite elements, which have a one-to-one relationship with the geometric finite elements of the respective geometric entity. It is important to note that phenomenon meshes have a reference element called the **PhenReferenceElement**, which is associated with the GeomReferenceElement and is capable to provide that part of the integration data, which depends on phenomenon data intrinsic data. This part is typically intrinsic since it does not depend on any external information. It comprises the values of the trial and test shape functions and their derivatives - up to a given order – at the integration points – provided by the respective GeomReferenceElement. All of those portions of data are encapsulated in the **PhenIntrinsicIntegData**. Based on what was just explained, it is natural that the PhenReferenceElement is the owner of the trial and shape functions, which are encapsulated in the **PhenShapeFunction**.

The information regarding the approximation of the vector field on each geometric finite element (from points to volumes) is stored in the **PhenElementData**. Therefore, this information is important if one wants to know the set of shape functions, which are non-zero on the finite element under consideration. This is the case when a weak form is requested to compute a vector or matrix at element level, since their size depends on the number of non-zero shape functions on the element and their values depend on which are the non-zero shape functions.

Interaction

During a simulation, the Computational Phenomenon objects are requested to perform some tasks by the solution algorithm. The sequence diagram in Figure 5-8 illustrates the interactions between some objects of the Computational Phenomena pattern after a request for the computation and assembling of a coupled quantity (specified by a code). The quantity is to be computed using a certain set of states from other phenomena (coupled phenomena) and assembled in a data structure. A phenomenon state is either a vector or a matrix, which has an identifier and is stored outside the Computational Phenomenon. The solution algorithm used in the simulation determines the meaning and number of states maintained for each phenomenon. Therefore, they are stored outside the Computational Phenomena under the responsibility of the Group, which owns the respective phenomenon. The retrieving of the states is made through the coupled phenomenon, which asks its Group to provide the right set of states.



SolutionAlgorithm : Algorithm
PhenA : Computational Phenomenon
PhenGraphRoot : PhenEntity
PhenRootWeakForm : WeakForm
CoupledPhen : PhenEntity
PhenEntChildren : PhenEntity
Contribute(code, structure, states)
Contribute(code,structure,states)
Contrib(structure,states)
RetrieveData(states)
RetrieveStates()
ReturnData()
ComputeAndAssemble()
Contribute(code,structure,states)
Return()
Return()
Return()
Return()
The same process executed by PhenRootWeakForm is recursevely performed by its children until the leaves of PhenGraph

Figure 5-8 Sequence diagram for Computational Phenomenon

When the solution algorithm (**SolutionAlgorithm)** wants to compute a certain phenomenon quantity and assemble it in a specified data structure (**structure**), it asks the Computational Phenomenon object (**PhenA**) to contribute and send information about the quantity code, data structure and identifiers of the states to be used by the coupled phenomena, through the command **Contribute (code, structure** and **states)**. PhenA, then, forwards the same message to the root of its PhenGraph (**PhenGraphRoot**). The PhenGraphRoot forwards the request **Contrib (structure** and **states)** to its weak form (**PhenRootWeakForm**) correspondent to the quantity code provided. The PhenRootWeakForm retrieves the needed data (states, and other relevant pieces) from its coupled phenomena (**CoupledPhen**). Next, the PhenRootWeakForm will compute the right contribution for each one of the PhenFiniteElments of the PhenGraphRoot's PhenMesh and assemble it into the data structure, provided as a parameter. Then, the PhenRootWeakForm initiates a recursive procedure in order for its children PhenEntities (**PhenEntChildren**) to perform the same operation it has just finished. The recursion continues until all the PhenEntities of the PhenGraph are reached. At the end of this process the desired quantity is computed and assembled and the SolutionAlgorithm takes over the control of the simulation.

5.2.6 Example of Usage

The problem describes the dynamics of a rigid body attached to an elastic beam, with a temperature dependent constitutive relation, where both are also submitted to thermal loads. Consider the geometry defined in Figure 5-9, consisting of two sub-domains Ω_1

and Ω_2. The physical phenomena defined therein are transient and include: linear elasticity with a temperature dependent constitutive equation in Ω_1; rigid body motion of Ω_2 (this body has a certain distributed mass density **?**$_M$) and heat transfer in Ω_1 and Ω_2. More details about this example can be found in Appendix A, section A.1.

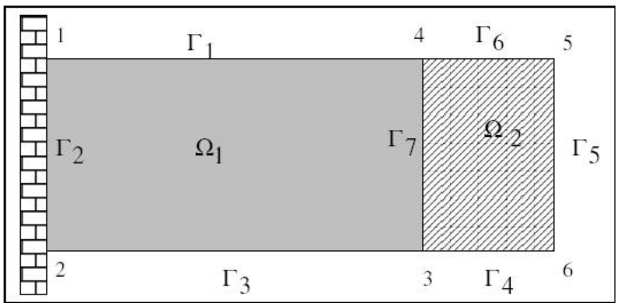


Figure 5-9 Whole domain of example

Geometry: as can be seen in Figure 5-9. There are 6 points, 7 curves and 2 plane regions defined for the problem geometry. Plane region Ω_1 is composed of curves Γ_1, Γ_2, Γ_3, Γ_7 and points 1,2,3,4. Plane region Ω_2 is composed of curves Γ_4, Γ_5, Γ_6, Γ_7 and points 3, 4, 5, 6.

Phenomena Context: composed of: elasticity, rigid body motion and heat transfer.

Phenomena: in the problem we can identify the following simulation phenomena:

- Heat transfer, phenomenon is represented by the temperature vector field T_1: $\Omega_1 \times \Re^+ \to \Re$
- Heat transfer phenomenon is represented by the temperature vector field T_2: $\Omega_2 \times \Re^+ \to \Re$
- Phenomenon represented by Lagrange multiplier vector field μ_q: $\Gamma_7 \to \Re$ (Lagrange multiplier in Γ_7, due to restrictions between T_1 and T_2)
- Elasticity phenomenon is represented by the displacement vector field $\mathbf{w}_1 : \Re^+ \to \Re^2$
- Rigid body motion is represented by rigid body displacement vector field $\mathbf{w}_2 : \Re^+ \to \Re^2$,
- Phenomenon is represented by the Lagrange multiplier vector fields m: $\Gamma_2 \times \Re^+ \to \Re^2$ and m$_\mathbf{f}$: $\Gamma_7 \times \Re^+ \to \Re^2$ (Lagrange multipliers in Γ_2 and Γ_7, respectively due to restrictions of $\mathbf{w}_1$, and between $\mathbf{w}_1$ and $\mathbf{w}_2$).

Simulation Regions: these are the regions where the phenomena are defined. The regions, where restrictions between phenomena are defined, are included as simulation regions. In the example we have:

- Ω_1, where elasticity and heat transfer are defined;
- Ω_2, where rigid body motion and heat transfer are defined;
- Γ_7, where restrictions between T_1 and T_2 and a restriction between $\mathbf{w}_1$ and $\mathbf{w}_2$ are defined;
- Γ_2, where a restriction involving only $\mathbf{w}_1$ is defined.

Below one can find the specification of the data for each phenomenon.

a) Phenomenon Specification:

Each one of the described phenomena has its own discrete vector field, geometric domain, and couplings with other phenomena, discrete weak form, and other relevant data. Furthermore, for all phenomena we define the following:

- Shape Functions (Test Functions and Trial Functions): a tool for providing the values of the shape functions and their derivatives up to a given order at the integration points;
- Phenomenon methods
  - Integration rule: a routine for providing integration points and respective weights with respect to the geometric reference finite element;
  - Geometric mesh generation method;
  - Phenomena mesh generation method;
  - Initial state;
  - Compute initial time step;
  - Compute next time-step.
- Restrictions (Dirichlet or other type)
  - Restriction between $\mathbf{w}_1$ and $\mathbf{w}_2$ on Γ_7 and $\mathbf{w}_1 = b.\mathbf{w}_2$;
  - Restriction $\mathbf{w}_1 = \mathbf{0}$, on Γ_2;
  - Restriction $T_1 = T_2$ on $\Gamma_{7.}$
- Phenomena Couplings
  - Between $\mathbf{w}_1$ and T_1 on Ω_1 (constitutive coupling the material constitutive relation depends on T_1);
  - Between $\mathbf{w}_1$ and the Lagrange multiplier m_f defined on Γ_7;
  - Between $\mathbf{w}_1$ and the Lagrange multiplier m defined on $\Gamma_{2;}$
  - Between $\mathbf{w}_2$ and the Lagrange multiplier m_f defined on Γ_7;
  - Between the phenomenon boundary condition for T_1 defined on Γ_7 and the Lagrange multiplier μ_q;
  - Between T_2 and the Lagrange multiplier μ_q defined on Γ_7.

b) Phenomena Boundary Conditions:

Boundary conditions for $\mathbf{w}_1$:

- On Γ_1 and Γ_3 there is a Neumann boundary condition with zero loads in each one of them. Thus, void sub-phenomena (which do nothing) are defined there.

- Dirichlet condition on $\Gamma_{7.}$ This was already considered as a restriction, which generated an extra phenomenon with vector field μ_f. However, w_1 became coupled with this Lagrange multiplier. Therefore a sub-phenomenon is defined on Γ_7 in order to provide for the quantities related to that coupling.
- Dirichlet condition on $\Gamma_{2.}$ This was already considered as a restriction, which generated an extra phenomenon with vector field μ. However, w_1 became coupled with this Lagrange multiplier. Therefore a sub-phenomenon is defined on Γ_2 in order to provide for the quantities related to that coupling.

Boundary conditions for $\mathbf{w}_2$:

- On Γ_i, i = 4, 5, 6 there is a Neumann boundary condition with zero load in each one of them. Thus, void sub-phenomena (which do nothing) are defined there.
- Dirichlet condition on $\Gamma_{7.}$ This was already considered as a restriction, which generated an extra phenomenon with vector field μ_f. However, w_2 became coupled with this Lagrange multiplier. Therefore a sub-phenomenon is defined on Γ_7 in order to provide for the quantities related to that coupling.

Boundary conditions for T_1:

- On Γ_2 there is a zero Neumann boundary condition (insulation). Thus, a void sub-phenomenon (which does nothing) is defined there;
- On Γ_1 and Γ_3 there are mixed boundary conditions (convection) prescribed. Therefore two sub-phenomena will be defined there;
- Dirichlet condition in Γ_7. This was already considered as a restriction, which generated an extra phenomenon with vector field μ_q. However, T_1 became coupled with this Lagrange multiplier. Therefore a sub-phenomenon is defined on Γ_7 in order to provide for the quantities related to that coupling.

Boundary conditions for T_2:

- On Γi, i = 4, 5, 6 there are mixed boundary conditions (convections) prescribed. Therefore, three sub-phenomena will be defined there;
- Dirichlet condition in Γ_7. This was already considered as a restriction, which generated an extra phenomenon with vector field μ_q. However, T_2 became coupled with this Lagrange multiplier. Therefore a sub-phenomenon is defined on Γ_7 in order to provide for the quantities related to that coupling.

Representation of Phenomena and Geometry through graph structures

Figure 5-10 presents the GeomGraph, which represents the supplied geometry where phenomena occur and which are composed of GeomEnitites. Note that the graph represents the problem geometry composed by surfaces, curves, and points presented in Figure 5-9.

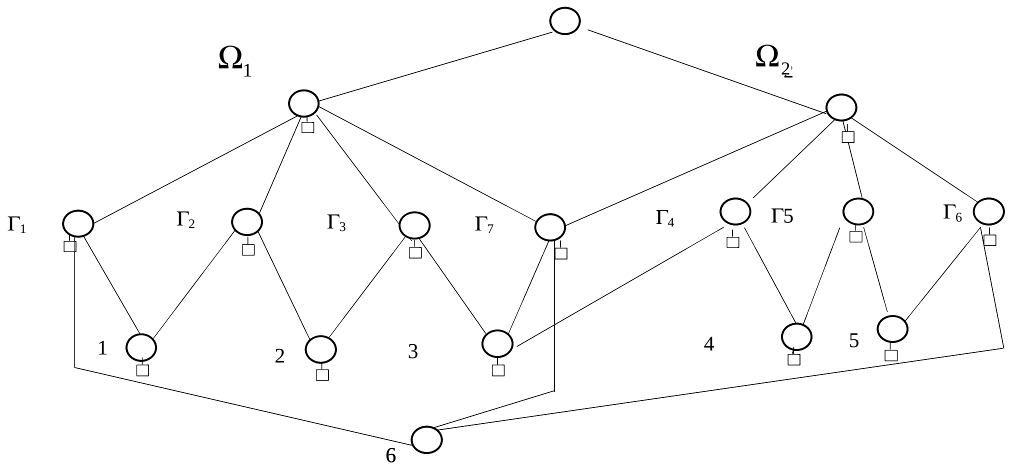
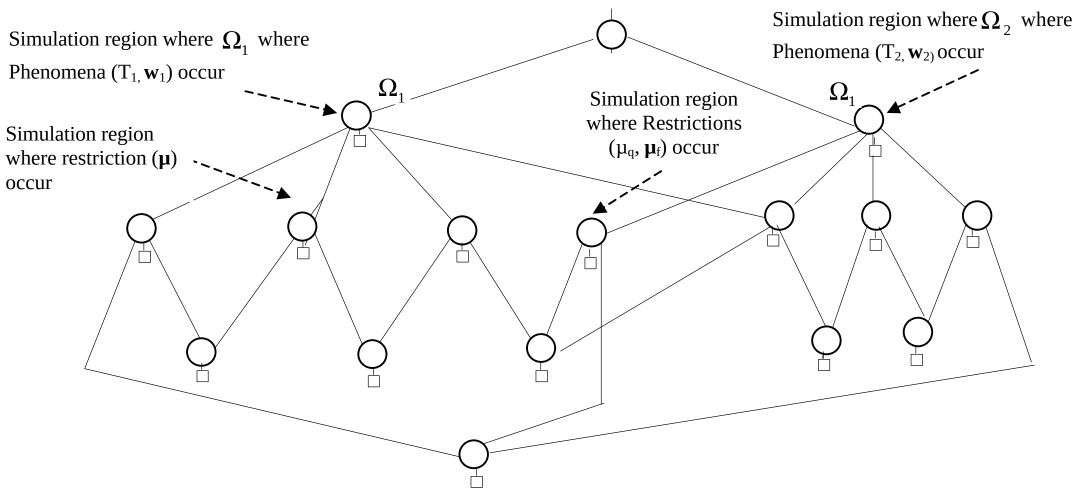


Figure 5-10 GeomGraph of the whole example geometry

Figure 5-11 shows the Simulation Regions and phenomena defined on them. Remember that a copy of the simulation region is given to each phenomenon defined therein, which becomes the GeomGraph of the phenomenon.



Simulation region where Ω_1 where Phenomena (T_1, $\mathbf{w}_1$) occur
Simulation region where Ω_2 where Phenomena (T_2, $\mathbf{w}_2$) occur
Ω_1
Simulation region where Restrictions (μ_q, $\boldsymbol{\mu}_f$) occur
Ω_1
Simulation region where restriction ($\boldsymbol{\mu}$) occur

Figure 5-11 Simulation Regions and their phenomena

The PhenGraph represents the phenomena controller responsible for the simulation data, described previously in Figure 5-6. As noted previously, the PhenGraph resembles the organization of the respective simulation region (represented by its GeomGraph). Each GeomEntity of the simulation region is associated with a PhenEntity, which represents a specific simulation data, which in turn implements the phenomenon in that GeomEntity.

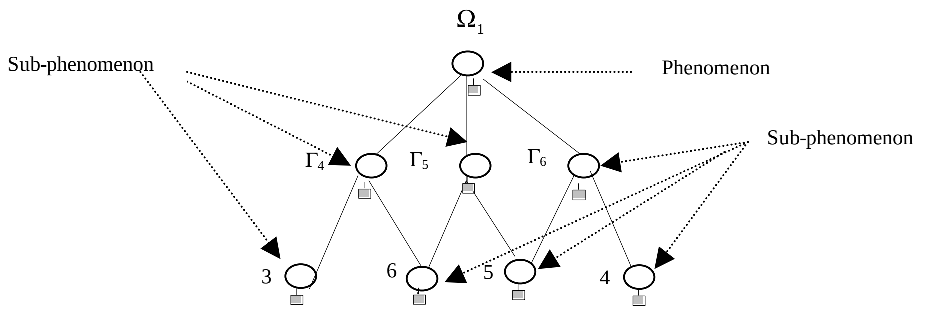


Figure 5-12 Example of a PhenEntity

Through space discretization and the use of the FEM it is possible to obtain the following semi-discrete equations (discretization in time is still to be done).

5.2.7 Consequences

Several forces were considered as relevant for the definition of a computational phenomena abstraction (see section 5.2.4). They included: levels of defined abstractions, degree of completeness of involved concepts, reliability of the concepts, possibility of data persistency, standardization of the model in the community, performance, and the management and automating of simulator building, etc.

We can observe that the proposed solution solved some of the following forces:

- Definition of higher levels of abstraction;
- Simplification on computational phenomena use and implementation;
- Completeness of the information provided was improved, however;
- Reliability of computer simulations is improved by the use of pre-defined models.
- Higher levels of reusability and maintainability were achieved.
- Persistence can be defined as structures for representing the underlying concepts were defined.

The definition of high levels of abstraction for the main concepts of problem data modelling were proposed to reduce the complexity and improve the correctness of the simulation to be developed. This was achieved by an adequate separation of concerns, such as the separation between the solution processes and the computations of the contributions of each phenomenon to the equations to be solved. Note that many pieces of information regarding the modelling of a phenomenon are used in different phases of the simulation. For instance, mesh generation methods are used in the pre-processing phase, while numerical integration methods are used in the simulation itself. However, both pieces are directly related to the numerical modelling of a phenomenon. Thus, many aspects of the structure, dynamics and relationships of the sets of data related to a phenomenon could be analysed in order to achieve high levels of abstraction with the resulting desired benefits. The abstractions used can reduce the involved complexity in the development of FEM simulators and their maintenance, and also give support to their automation.

For the independence of the higher levels of simulator programming, we separated the parts responsible for the solution processes and the parts responsible for the production of the vectors and matrices, at the finite element level, and their assembling in global entities. This separation becomes possible when the machinery for the production and assembly of those vectors and matrices is provided and is made flexible enough to accommodate for the requirements of large classes of solution processes.

Some negative forces can also be identified such as performance. If phenomenon objects, for the computations of matrices and vectors, do not provide special tools performance is expected to be very low, due to the high level of abstractions involving and integrating many concepts.

5.3 FEM Simulator Skeleton Pattern

The FEM Simulator Skeleton pattern supports the conception of FEM simulators. This pattern makes possible to separate complex procedures from simpler ones and strongly re-usable software components from less reusable ones. Furthermore, it opens the way to automatic programming of FEM simulators for coupled phenomena. One immediate benefit is the enhancement of reusability.

The FEM Simulator Skeleton considers four levels of computation for FEM simulator conception. It supports abstractions for different phenomena coupling in a single strategy, identifying which parts can be more reusable than others and proposes a hierarchical and modular solution. The pattern also suggests a physical phenomenon abstraction, called here Computational Phenomenon, which was described previously in section 5.2.

5.3.1 Name

The FEM Simulator Skeleton, which means a pattern for modelling FEM simulators based on algorithm skeletons for coupled phenomena.

5.3.2 Context

When a designer defines a computational model for a mathematical formalism, using FEM in the context of coupled phenomena, he/she has to deal with problems such as data dependence and sharing. Such issues are not trivial to treat in a homogeneous way because they are strongly dependent on the specific problem being considered. Thus, it becomes difficult to provide reasonably high levels of abstractions, which could represent the main components, properties, relationships, and operations involved. Without that, even when making use of sophisticated FEM libraries, the tasks involved in building and assessing the performance of new methods could become very costly and time consuming due to lack of modularity and reuse. In addition, as far as we are concerned, there is no standardized solution for the control of coupled phenomena simulations, making the integration of reusable components a very difficult task in this context.

In this pattern, we are concerned with the conceptualisation of the Simulation Process. We assume that the simulator building and assembling will be based on a variable designer data model, which describes: the initial scenario, algorithm skeletons and numerical methods, phenomena, and geometry. The initial scenario defines the class of problems that the simulator will be able to tackle in a broad sense, as was described in previous chapters.

5.3.3 Problem

How a complex simulator for coupled multi-physics phenomena based on FEM can be structured in such a way that it guarantees a high level of reuse and modularity?

5.3.4 Forces

The FEM Simulator Skeleton pattern aims at solving forces related to high costs in complex simulation systems development, particularly in the direction of complexity management and software quality achievement. Nevertheless, this pattern also considers the automatic articulation of solution strategies for coupled multi-physic phenomena and their possible replacement by other articulations. In the sequence we describe the forces involved in the context of FEM coupled phenomena simulator modelling:

- High complexity: there is a lack of standard abstractions to help the simplification and organization of complex structures of data and code related to coupled phenomena simulations in the FEM context. The relationships among phenomena are strongly problem-dependent and solution algorithm dependent.
- Reusability: numerical experiments are complex constructs, based on pieces of information such as strategies, auxiliary methods, and other pieces of data. They can be very reusable for large classes of problems.
- Adaptability: due to the frequent improvement of numerical methods or due to the need of comparing different methods, the simulator architecture must be adaptable (to some extent) to support the required modifications without heavy reprogramming.
- Strategy Independence: in order to allow the designer to specify the simulator features and strategies, there must exist flexibility in building different solution strategies.
- Integrability: there is a need for an application/routine that is able of monolithically solve a specified set of coupled phenomena. Some problems simply do not allow for an independent solution for each phenomenon.

Furthermore, whenever different software components have to be used together for the simulation of coupled phenomena (for instance, in a partitioned way), problems concerning data transfer and integration frequently appear.

5.3.5 Solution

The main structure of the pattern for representing a general FEM simulator is composed of Simulator, Block of Groups, Group of Phenomena, Phenomenon, Algorithm Skeletons and MathMethods, see Figure 5-14. The FEM Simulator Skeleton pattern suggests a FEM simulator algorithms organization with four levels of computational demands: Global Skeleton, Block Skeletons, Group Skeleton, and Phenomenon. These levels were defined due to the high number of repeated (similar) structures and the degree of reusability of the involved algorithms (see example in section 5.3.7).

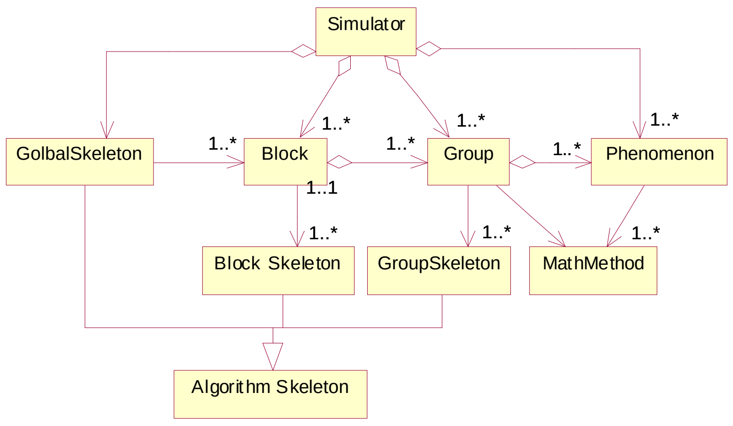


Figure 5-13 Participants of the Simulator Pattern

Participants

The FEM Simulator Skeleton pattern is composed of the following participants:

- *Simulator* represents a class of possible simulations and it is responsible for the control of the main process flow; thus, it maintains the core of simulation through the Global Skeleton, which is stored in the Kernel.
- *Algorithm Skeletons* are algorithms described by the simulator designer, corresponding to one of the levels of computation (Global, Block, Group), using the pattern-defined abstractions.
- *MathMethod* is a routine with a very specific purpose and is used by either Algorithm Skeletons or encapsulated procedures inside a Phenomenon. For instance, MathMethods are defined by: numerical integration, mesh adaptation, error estimation, and other tasks.
- *Kernel*: main part of the simulator, which is related to the global Skeleton and which also controls the simulator workflow.

- *Global Skeleton* is the highest level of the solution scheme and it articulates the action of all Blocks contained in the Kernel. It is supposed to be strongly reusable.
- *Block* contains a set of Groups of phenomena. Each Block has a set of skeletons called Block Skeletons. More than one block is justified, for instance, in the case where a problem is partitioned into the solutions of separate sets groups of phenomena.
- *Block Skeletons*, where the Groups are required to perform a certain number of categories of procedures (for instance, partitioned - staggered - solution procedures involving groups of phenomena). When a Group is asked to execute a category of procedures (for instance, to compute a solution for its group of phenomena), it executes a very specific algorithm, which is a member of that category. Block Skeletons are supposed to be very reusable.
- A *Group* contains a set of phenomena, which are going to be solved monolithically. A Group is provided with a set of Group Skeletons.
- *Group Skeletons* represent very specific procedures. Due to their problem- and method-specific definition and organization, the Group Skeletons are the least reusable amongst all Skeletons. Nevertheless, they may be implemented in such a way that they become capable of considering a varying number of phenomena, depending on the requirements from the simulation design.
- A *Phenomenon* represents a complex system composed of data and tools. Its primary responsibility is to provide the contributions of each phenomenon to a Group System of equations to be solved in each instant of the solution process. This level is the place where the couplings and other processes of data sharing and dependence are considered in the construction of the required vectors and matrices. It is the lowest level of the procedures in the solution schemes and thus, it represents a tremendous effort in terms of programming, testing, and validation. Therefore, the reusability of the tools located in the classes, which compose what we call a Phenomenon, is fundamental to saving time and cost whenever one is programming new simulations.

Levels of Computation

The four levels of computational demands (skeletons and methods) are detailed below:

- *Global Skeleton* is the first level of computation and represents the global algorithm skeleton (the core of the simulator). The global algorithm skeleton articulates the procedures involving all blocks. The procedures here deal with a high level of simulation execution, such as time loops, and adaptive iterations, etc. It also includes general requirements such as asking the blocks to obtain the block solution or to perform an adaptation procedure. There is no need for matrices and vector manipulations at this level. The building of a Global Skeleton depends on a series of decisions about the whole classification of the simulation. A Global Algorithm Skeleton is unique for each simulator, but may be replaceable, producing another simulator. Global Algorithm Skeleton is the procedural structure representing the algorithm to be performed with demands defined at a higher level. It does not refer directly neither to a Group of phenomena nor to any phenomenon.

- *Block Skeletons* are necessary in order to articulate the Groups of Phenomena in the execution of tasks demanded by the Global Skeleton. Each block has a set of skeletons (Block Skeletons), which satisfies the demands from the Global Skeleton by decoding them into demands for the groups in a previously defined order. A simulator may have a Block Skeleton changed without requiring the modification of the simulator's Global Skeleton. Nevertheless, a well-designed Block Algorithm Skeleton is also very reusable and it is not supposed to be replaced even in the case of very severe changes in the solution algorithm at the level of the phenomena Group. The Block Skeleton defines solution procedures such as iterations in the case of operator splitting solution strategies (which involve all Groups), iterations in the case of non-linear solvers (involving one or more Groups), etc. It also transfers directly to its Groups some of the demands coming from the Global Skeleton (time step estimation, error estimation, etc.) and possibly post-processes the output from the Groups.

- *Group Skeletons* are necessary in order to articulate the Phenomena in the execution of tasks demanded by the Block Skeletons. A Group is provided with a set of Group Skeletons, which represent very specific procedures and may not be very reusable. Its purpose is to encapsulate the parts from the solution scheme, which are specific to the particular solution method being used for a group of phenomena. Usually, the more reusable parts of the solution scheme are best located either in a Block Skeleton or in the Global Skeleton. In the Group Skeletons, the quantities produced by the Phenomena Skeletons are manipulated in the way required by the solution method, which characterizes the Group. Thus, the Group becomes specialized in the solution of any subset of its set of possible phenomena; hence, all vectors and Matrices used in the solution are located in the Groups. The Group also needs to have knowledge of its Phenomena couplings, whenever building coupled terms. This is because the coupled terms have been built, possibly using an already computed discrete vector field (possibly related to another group), which should be appropriately defined. Frequently, Group Skeletons make use of MathMethods, whenever there is a task, which can be encapsulated representing either a reusable or a replaceable procedure (solution of an algebraic system of equations, for instance).

- *Phenomenon Procedures* represent the lowest level of all procedures in the simulation and are related to all possible contributions its Phenomenon can provide to any solution scheme. Starting from the computation of the Global Skeleton and going through the two other levels of articulation, what remains to be defined are the contributions of each phenomenon to its Group solution scheme in a uniform parameterised way. The phenomena classes will be composed of phenomenon data and a group of numerical methods (MathMethods), which are replaceable (i.e. can be modified by the users through input data, such as integration rules, for instance).

Interaction

We can summarize the pattern's major interactions in the following way: (a) the Global Skeleton articulates the procedures involving all blocks; it does not make any requirements directly neither to a Group of phenomena nor to any Phenomenon; (b) the Block Skeletons then define the activities of the groups; (c) the Group Skeletons in turn articulate the phenomena in their computations, that is represent how the phenomena will be solved together. This produces cleaner and more reusable Global and Blocks algorithm Skeletons, leaving to the Groups algorithm Skeletons the responsibility of defining the specific problem dependent (non-reusable) procedures of the whole solution algorithm.

5.3.6 Applicability

The proposed pattern has great applicability in FEM simulation modelling especially when the following situations are frequent:

- Several phenomena defined in the same geometric region, with either different meshes and different adaptation criteria or sharing meshes and other data;
- Interchange of data between phenomena is very frequent (data dependence);
- Assessment of solution quality may be different and sometimes interdependent (error estimation, adaptation, and approximation properties) from one phenomenon to another;
- The desired solution algorithms articulate separate groups of phenomena and those groups, in turn, consider sets of phenomena in the computation procedures (as it is the case in operator splitting schemes).

5.3.7 Example of Usage

In this sub-section, we show the application of the pattern to a general scenario of a simulator model, which is described in Appendix A. Next we will detail the pattern application for the proposed simulator scenario. Then, some considerations related to the pattern application are presented. Finally, we show an example of a problem that can be solved by the defined simulator.

Usually, it can be observed that an algorithm defined for the solution of a problem by the FEM method has repeated (similar) structures. Thus, in the pursuit of a high degree of reusability, four levels of demands in the algorithm were devised: Global Skeleton, Block Skeleton, Group Skeleton, and Phenomena procedures. In the Block Skeleton we will assume that N_g^r is the number of groups for the rth -block. In the sequence we will present the algorithm Skeleton and the Global Skeleton.

Figure 5-14 shows the Global Algorithm Skeleton for the proposed Simulator. As it involves, for example, transient phenomena it includes tasks to compute initial time steps for blocks and the computation of the next time step.

```
I.) From Blocks i = 1 until 2
        I.0) Retrieve initial state for Block i
        I.I) Compute initial time step $Dt_I$ for Block i
        I.II) Compute initial auxiliary data for Block i
II) Compute initial D t = min $_{1 £ i £ 2}$ {$Dt_i$} and set time instant $t_1 = 0$
III) While $t_1$ £ $T_{max}$ do:
        III.0) Set $t_0 = t_1$ and $t_1 = t_0 + D t$
        III.I) For Block i = 1 until 2
              III.I.0) Solve for Block i
              III.I.I) Compute next time $Dt_i$ for Block i
        III.II) Compute next time step D t = min $_{1 £ i £ n}$ {D $t_i$ }
        III.III) Continue with time iteration
IV) End of the simulation
```

Figure 5-14 Global Algorithm Skeleton

Figure 5-15 details the Block Skeleton for the proposed Simulator. It is composed of sub-skeletons that implement for example: the initial state for the Block (I.0), the solution for the block (III.0), etc.

Observe that the Block Skeletons articulate the groups in a very simple way, almost only sending to the groups the requests made by the Global Skeleton. Nevertheless, it should be noted that the decision of providing an iterative scheme involving the Groups was made and defined by the Block Skeleton. In this sense such a procedure is transparent to the Global and Group Skeletons.

The Group Skeletons are subtler in what concerns the articulation of their phenomena for providing the demands of the original solution algorithm. The detailed description of Group Skeletons is beyond the objectives of this example. However, it can be seen from the algorithm that each Phenomenon should provide the group with matrices and vectors for assembly.

```
Is-Br) Initial State for Block r (see (I.0))
            Is-B$^r$.0) For **i = 1** until $N_g^r$
                        Is-B$^r$.0.0) Ask Group i to compute Initial state for its phenomena
It-B$^r$) Initial time step for Block r (see (I.I)):
            It- B$^r$.0) For **i = 1** until $N_g^r$
                        It- B$^r$.0.0) Ask Group i to compute Initial time step Di
            It-B$^1$.I) Set Dt$^1$ = min $_{1£ i £ N^1_p}$ {D$_i$ }
Id-B$^r$) Compute initial auxiliary data for Block r (see (I.II))
            Id-B$^r$.0) For i = 1 until $N_g^r$
                        Id-B$^r$.0.0) Ask Group i to compute its auxiliary data.
Sl-B$^r$) Solve for Block **r** (see (III.0))
            Sl-B$^r$.0) Initialise iteration state **k = 0** for Block **i**
            Sl-B$^r$.I) Set **k = 0**. While convergence for Block **r** is not achieved, do:
                        Sl-Br.I.0) Compute the (k+1) th-solution  based on the k th-solution for  Block r
                        Sl-Br.I.I) Compute error between the solutions k and k+1 for Block r
                        Sl-Br.I.II) Compute auxiliary data for next step and increment k = k+1
            Sl-B$^r$.II) Accept last solution from iteration loop for Block **r**
Sk-B$^r$) Initialise iteration state k = 0 for Block r (see (Sl-B$^r$.0)):
            Sk-B$^r$.0) For i = 1 until $N_g^r$
                        Sk-B$^r$.0.0) Ask Group i to initialise iteration state k= 0
Sl-B$^r$) Compute the (k+1)$^{th}$-solution from the k$^{th}$ -solution for Block r (see (Sl-B$^r$.I.0)):
            Sl-B$^r$.0) For i = 1 until $N_g^r$
                        Sl-B$^r$.0.0) Ask the Group i to compute its (k+1)$^{th}$ -solution from its   k$^{th}$ –solution
Er-B$^r$) Compute error between the (k+1)$^{th}$ -solution and the k$^{th}$ -solution for Block r  (see (Sl-B$^r$.I.I)):
            Er-B$^r$.0) For i = 1 until $N_g^r$
                        Er-B$^r$.0.0) Ask Group i to compute its error Ei$^k$
            Er-B$^r$.I) Compute Block error E$^{r,k}$ based on the Group errors { Ei$^k$} 1£ j£ $N_g^l$
Ad-B$^r$) Compute auxiliary data for Block r at k$^{th}$ -iteration (see (Sl-B$^r$.I.II)):
            Ad-B$^r$.0) For i = 1 until  $N_g^r$
                        Ad-B$^r$.0.0) Ask Group i to compute its auxiliary data.
As-B$^r$) Accept last solution obtained in the iteration for Block r (see (Sl-B$^r$.II)):
            As-B$^r$.0) For i = 1 until $N_g^r$
                         As-B$^r$.0.0) Accept last solution obtained for Group i and store it.
Nt-B$^r$) Compute next time step for Block r (see (III.I.I)):
            Nt-B$^r$.0) Ask group i to compute next time step D$_i$
        Nt-B$^r$.I) Set $\Delta t^r = \min_{1\le i\le N_g^r} \{\Delta_i\}$
```

Figure 5-15 Block Skeleton for any Block

In the above *Vec* is a given vector and *a* and *b* are given scalars. Each one of those quantities that a phenomenon offers to its Group may depend on vector fields from other phenomena (either from its Group or not). It is the responsibility of the current Group to indicate: (i) the quantity to be computed by its phenomena and (ii) in the case of coupling, what is the vector field state (either from the current Group or not) that the coupled phenomena should use in order to provide what is needed for the computation of the coupled quantity. With such an organization, demands to the Phenomena become very uniform, making them extremely reusable.

A final remark is related to the fact that this pattern was made possible by the way the data and tools are built in the Phenomenon level. It is in this level that data dependence and sharing between phenomena are defined, leaving the Global Skeleton and the Block Skeletons free from those details. The Group Skeletons are the agents responsible for mapping the requirement of a phenomenon for quantities from other phenomena to the actual quantities, which are stored either in the current Group or in other Groups.

5.3.8 Considerations

This pattern considers that the class of problems, which define the applicability of a simulator, can be defined in a somewhat clear way. For instance, considering only its Global Skeleton, the Simulator built in the example (first sub-section of section 5.3.7) is capable of solving simulations in the class of dynamic problems with neither adaptation nor error estimation. Now, considering its Block Skeletons, it is capable of solving only linear (or very mild non-linear) problems with Dirichlet type restrictions and using a split stabilized methodology. Those restrictions may involve one or more vector fields. The Group Skeletons are very specific to the solution scheme used and even slight modifications may cause the necessity to redesign and reprogram them. As noted, the **couplings** and other processes of **data sharing** and **dependence** are considered at the phenomenon level leaving the Global and Block Skeletons free of having to consider them. Since Group Skeletons are the least reusable; they may (and frequently do) deal with specifying the right quantities that a coupled phenomenon should retrieve from its own Group. This reflects on coupling between Groups, which has been described earlier and is related to the specifics of the solution methodology being used by the Group.

5.3.9 Example of Simulator Applicability

An example of a problem that can be solved by the defined simulator is described in Appendix A – example 1 and 2. Is composed of two sub-domains Ω_1 and Ω_2. The physical phenomena defined therein are (transient state): linear elasticity with temperature dependent constitutive equations in Ω_1; rigid body motion of Ω_2 and heat transfer in Ω_1 and Ω_2. The proposed simulator will build the global linear system related to all the mesh elements, for each phenomenon, and solve this system. For the present example of problem formulation, to be applied to the defined FEM simulator we can consider the following:

- Groups: group 1, phenomena represented by vector fields T_1 and T_2 (heat transfer in Ω_1 and Ω_2); group 2, phenomena represented by vector field μ_q (Lagrange multiplier in Γ_7, due to restrictions between T_1 and T_2); group 3, phenomena represented by their vector fields w_1 and w_2 (elasticity in Ω_1 and rigid body motion in Ω_2); group 4, composed of the phenomena represented by their vector fields μ and μ_f (Lagrange multipliers in Γ_2 and Γ_7, respectively, due to restrictions in w_1).
- Blocks: block 1, composed of groups 1 and 2; block 2, composed of groups 3 and 4.

5.3.10 Consequences

It is worthwhile observing that a FEM Simulator Skeleton pattern is not restricted to a given implementation of Blocks, Groups and Phenomena. Their abstract behaviour and interaction are independent of a specific implementation. When dealing with the building of a specific Simulator, the implementation of the Global and Block Skeletons should reflect the needs for the solution of a large class of problems, which constitutes its strategy. Thus, each Simulator built, based on the proposed pattern, should be capable of solving completely different problems, defined by completely different geometries and considering completely different sets of phenomena, provided that the problem is still within its applicability range.

Forces solved by the pattern

The FEM Simulator Skeleton pattern consider:

- Higher levels of abstraction for the main concepts of FEM Simulation Skeleton pattern modelling, giving support for the reduction of complexity and correctness of the systems (simulators) to be developed.
- Higher levels of hierarchical modularity for the system process organization, by the use of global skeletons, blocks and group skeletons.
- A solution, which may consider monolithic, coupled phenomena simulation.
- The higher levels of code reusability are found in the Phenomena, Global and Block skeleton structures, followed by Group of phenomena. The less reusable is the group of phenomena, because it is the location more sensitive to modifications, whenever changes in the numerical method and type of simulation are desired.
- Reliability of the computer-generated predictions is considered by the use of pre-defined strategies, numerical methods and templates.
- A higher level of maintainability is supported due to the defined modularity in the simulator modelling, that is, the separation of the different levels of computation.

5.3.11 Negative Consequences

In the FEM Simulator Skeleton pattern some negative consequences can be identified: the model builders require special training, that is, the designer must understand the proposed abstractions; designers will only achieve higher levels of reusability if they know how to articulate their strategies and problems; simulator performance can decrease due to the extra levels of abstraction imposed. However, one may notice that the number of calls to Blocks, Groups and Phenomena are very small.

5.3.12 Forces unsolved by the pattern

Some forces are still not solved or not even treated in the present work:

- Automatic programming: this is desired due to the great volume of code that must be reprogrammed in a single application of coupled phenomena.
- Expertise level: there are multiple standard situations and states, which are neither assisted nor guaranteed.
- Performance: generally the simulations are very computer time consuming. So the performance must be taken more seriously into consideration.
- Scalability: simulations frequently require a large volume of data, which can be partitioned and processed by many processors in a distributed memory environment. So, it is important to allow the increase of processors if required.
- Portability: the simulations code should have high levels of reusability. So, it is important for it to be portable, that is, to be used with different computational environments, in order to take advantage of different existing expertise defined therein. Specialists frequently interact in building multi-physics simulations.
- Reliability: computer-generated predictions are of great concern to specialists. They help, for example to detect critical problems. Reliability of the system is very important.
- Simulation Pre-processing: pre-processing of input data is an important task, since the simulator structures require complex mapping of the real input data. In addition, the data structure may ease the burden on the global algorithms complexity, concerning data sharing and data dependence between different phenomena.
-

5.3.13 Related patterns

The authors did not find any pattern that was specific to algorithm hierarchical modularisation for simulations based on FEM. There are some works, however, which present some level of abstraction and modularisation [LAN97, LAN99, PWC97]. Specifically, in the simulation of coupled phenomena based on the FEM, there are some works under development [LSA01, LSR02b, LSA02b], but not yet in a pattern form.

5.3.14 Known uses

Due to tremendous ongoing activity in the fields of application of the FEM, there is a need for tools, which could help the development of simulators with a high reusability degree in both the academic and industrial worlds. The expected users of this pattern are scientists and engineers who already deal with development of FEM codes in some level, or, at least, have a basic knowledge of that method.

5.4 GIG-Pattern (Generic Interface Graph)

The use of workflow technology helps the development of more flexible and versatile computation strategies. So, workflow management systems are a relevant support for large classes of business applications, and many workflow models as well as commercial products are currently available [CFM02]. While the comprehensive availability of tools facilitates the development and the fulfilment of customer requirements, workflow applications still require simple, generic and adaptive solutions for the complex task of rapid development of effective applications, in particular when complex domains are involved.

The Generic Interface Graph for process control (GIG-pattern) was developed after observing that many numerical algorithms showed the very same organizational structure when trying to achieve process reuse and flexibility for the adaptation of new strategies. Such organizational structure in turn allowed for an abstraction, which resulted in the GIG. As will be seen, in section 5.4.9, it is possible to devise frameworks to use the GIG pattern in order to implement different processes in a very flexible and automatic way.

The GIG-pattern describes an abstract workflow solution, whose purpose is to provide expressiveness and adaptability through simplified workflow programming, control and use [LSV03a]. Another GIG motivation is to maintain predefined algorithmic structure, which means that the translation from an algorithmic language representation of the processes into a computer one must be as direct as possible. This is important because, the achievement of similarity between the way the programmer has its algorithmic code organized and the implementation of it can bring simplification in further required changes. Also, sometimes, developers need solutions that do not make restrictions on the scale of the process, that is, which need a mixture of small-scale processes (that execute within applications) and large-scale processes (that execute on top of applications). Usually this situation happens when designers are also the programmers.

As a workflow pattern, GIG provides for the separation of process logic from task logic, which is embedded in user applications, allowing both to be independently modified and the same logic reused in different cases. The GIG-pattern considers

features related to run-time control functions [WMC95], which manage the workflow processes and the order of the various tasks.
This pattern was devised from the experience obtained during the implementation of several simulators in the FEM context. Researchers of the Mechanical Engineering Department – UFPE found the need to organize their code in a way that was easier to adapt to new strategies and also to allow process reuse. The GIG pattern was the result of providing an interface for process control dealing with the specific requirements mentioned.

This pattern's description is organized as defined previously. It includes some variants that can extend the pattern and an example of the FEM simulators context.

5.4.1 Pattern Name

Name: GIG-Pattern, Generic Interface Graph for process control.

5.4.2 Context

Many domain specific users, like scientists and engineers, still program in a procedural style. The reasons are many. Complex numerical systems usually make use of many different pre-built auxiliary packages (like numerical integrators, solvers for non-linear and linear systems of algebraic equations, etc) that have their procedures described in algorithmic language. Therefore, the majority of the work is related to making the modules compatible in a monolithic architecture, which resembles the structure of the algorithm. This is a strong force that drives those users towards the procedural style.

During the development of a software system, those developers need functions that help them to organize their logical processes and related tasks, in a way that makes easier its future adaptation to new solutions and for the reuse of software components, avoiding heavy reprogramming. We have repeatedly noticed that many numerical algorithms have exactly the same organizational structure. This structure comes from the procedural style of the algorithm representation and can be identified as a directed acyclic graph. This observation can lead to the definition of a workflow pattern as the one we are describing.

5.4.3 Motivation Example

Consider, for example the case of a mesh generation algorithm. A mesh can be described as a partition of a geometric domain into simple geometric entities (triangles, tetrahedral, hexahedral, etc) called geometric finite elements (or simply elements). In Figure 5-16 the algorithm for a particular mesh generation is presented, which, given a plane straight-line graph (PSLG), generates a mesh of triangles.

```
I.  Data input (PSLG)
II. Generate the bounding box for the PSLG
III.Build the initial mesh of the bounding box
IV.For each point in the PSLG do
IV.I.  Insert point
IV.I.I.  Find elements affected by the new point
IV.I.II. Eliminate those elements obtaining the affected region (AF)
IV.I.III Build new elements from the new point and boundary of AF
        Find a line of the PSLG such that it is not an edge of any triangle
        (negative line)
While there still is a negative line do
VI.I Compute the middle point of the line
VI.II insert middle point (see IV.I)
VII. Eliminate those triangles, which have any point of the bounding box as one of their vertices.
VIII.Data output
```

Figure 5-16 Mesh Generation Algorithm

This algorithm can be represented using the graph structure presented in Figure 5-17. Observe that there are fifteen sub-routines, including the driver (which executes the procedures I- VIII). This graph structure can be represented in a GIG-pattern (see Figure 5-18). Each one of those processes can be encapsulated in an object of a class, representing a node of the graph. The proposed pattern describes it as a derivation of a base class called *AlgthmNode.*

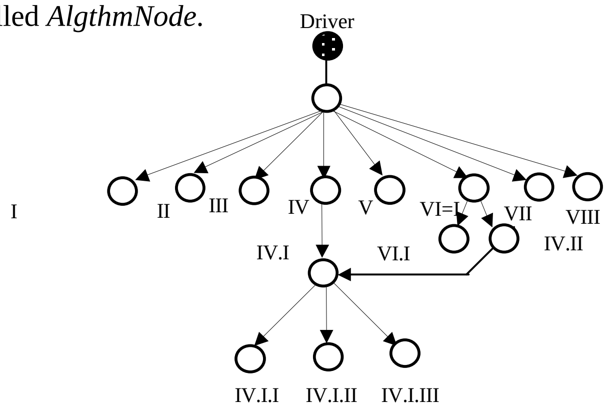


Figure 5-17 Mesh Generation Graph

Observe that there are many different ways of performing each one of the tasks described in the above algorithm. For instance, *IV.I.I find elements affected by the new point* concerns a search method in a geometric database of triangles, looking for a triangle whose circumscribed circle contains a given point. There are a lot of search methods available in the specialized literature, each one with its advantages, disadvantages and dependence on special data structures. Replacing the current method by a new one will not affect any other place in the graph.

Entire branches can also be changed as well. For instance, the process *IV.I. insert point*, can be changed by plugging in another method to perform that task. That means that all subsequent processes (children nodes) will also be changed. Besides the severity of the change in the methods needed by the algorithm, all substitution work can be automatically performed.

Then again, the Data Domain of this problem can be decomposed in such a way that all *AlgthmNode* objects (subroutines) will have access only to the data it needs. For instance, the process *III. Build an initial mesh for the bounding box* will need the bounding box and will build the initial mesh, which will be stored in a place in order to be accessed by other nodes. That decomposition will give rise to the classes derived from AlgthmData. The whole set of data pieces depend on the geometric data structure used by the developer. For instance, it can be seen that some structures have to be present: (a) PSLG (accessed by I, II, IV and V); (b) bounding box (accessed by II, III and VII), (c) mesh (accessed by III, IV.I.I, IV.I.II, IV.I.III, V, VII and VIII), (d) auxiliary data (many, it depends on the designer). All those pieces of data will be encapsulated in objects of classes derived from *AlgthmData,* see Figure 5-18.

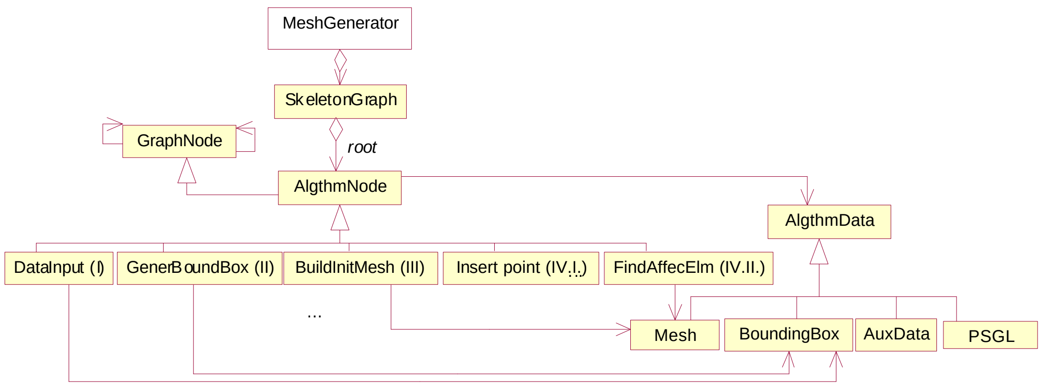


Figure 5-18 Application of the GIG structure in Mesh generation algorithm

In this example, the mesh generation process is the controlled workflow, see Figure 5-18. This process includes information about constituent tasks (represented as the processes (I to VIII). The mesh generation process has requirements related to modularity and exchange of sub-routines, since it has specific parts that have several kinds of implementations, which can be exchangeable.

5.4.4 Problem

How to guarantee simplicity in the separation of process logic from task logic, during the development of complex systems, while maintaining solution independence, reuse of processes and the predefined algorithmic structure?

5.4.5 Forces

With respect to the defined context, there are different forces, which lead to different solutions. Some of these forces are:

- Maintaining predefined algorithmic structure;
- Simplicity in the process definition;

- Support for different levels of granularity of the defined processes;
- Domain independence;
- Dynamic change of workflow processes;
- Reduction of error occurrences in the coupling of processes;
- Reuse of processes;
- Parallelism and processes synchronization;
- Workflow execution performance;
- Exploring existing expertise of domains of knowledge.

The following discussion analyses some of these forces, in order to identify how they are pulling against each other. GIG tries to resolve some opposing forces in the workflow definition context.

When trying to maintain the predefined algorithmic structure, the definition of some sub-process could generate pieces of code that are not easily changeable, because they are monolithically defined as a block of code. Conversely, refined levels of process partitioning can provide a process definition at statement level, eliminating existing abstractions (such as blocks or modules). Domain independence and dynamic change of process requires abstractions such as polymorphism and encapsulation, which are not present in a procedural style (the predefined algorithmic structure).

The guarantee of simplicity in process definition can be one method of avoiding errors and stimulate the pattern use. The reuse of already developed and tested processes helps in the simplification of process definition, similar to the possibility of reusing entire solutions. However, the reuse of processes can also reduce the simplicity due to the need for extensions of classes or configuration. Some other opposing forces to simple process definition are: the guarantee of domain independence, which makes the process definition more complex; also, to allow the definition of processes parallelism and synchronization the programmer has to deal with extra levels of complexity. Simplification can be compromised when parallelism is required for increasing performance.

Process reuse improves reduction of errors once pre-tested software is incorporated. Refined levels of granularity, in process definition, provide higher level of tangibility in the number of processes to be controlled, increasing the reuse of processes. The guarantee of domain independence also increases the number of reusable process.

Domain independence, avoiding non-monolithic solutions, makes the application of the workflow solution possible to different applications, improving its reuse. However, in these cases the existing expertise of a knowledge domain cannot be appropriately explored to improve the solution. Furthermore, synchronization and parallelism improve in one-way domain independent applications supporting the required functionality to existing applications.

The dynamic change of workflow processes improves the solution power. However, gives the programmer the responsibility and complex task of making a suitable division of code and data, for further exchanging to be pertinent. The reuse of process is fundamental when the user has to change an existing one for another one, which is already tested and classified. Maintaining the pre-defined algorithmic structure does not help domain independence because it does not provide, for example, encapsulation.

Parallelism and processes synchronization are very relevant to allow system optimisation and higher levels of control. The refined levels of granularity, in process definition, can allow a more precise level of parallelism definition. The dynamic change of workflow and the reuse of process increase the synchronization power, in process exchanging. Synchronization and parallelism improve in one-way the power of the dynamic change of process (identifying which process are independent or the dependence order). Conversely this can increase the complexity of the changing of processes.

Maintaining the predefined algorithmic structure can sometimes improve performance due to the direct application of some available optimised code; parallelism also improves performance, since it allows simultaneous execution of process. Alternatively, simplicity of process definition can decreases the performance, when it eliminates, for example, the possibility of parallelism definition. The guarantee of domain independence can also decrease performance once the existing expertise cannot be appropriately explored. Other forces, which compromise performance due to the need of extra verification and controls, are: refinement level of the granularity of process definition; dynamic process exchange; control of errors, reuse of process, and synchronization.

5.4.6 Solution

GIG can be described as a workflow solution [WMC95]. GIG follows the object-oriented style for modelling and programming. For simplicity, reasons of use and easy correctness verification, GIG implements a restricted direct acyclic graph (DAG) [LSV03a].

Participants (Structure)

The GIG structure is presented in the UML diagram in Figure 5-19.

The GIG pattern is composed of the following participants:

- *GraphNode*: this is an abstract class that implements low level operations related to the interoperability between graph nodes. It controls the relationship between workflow tasks.

- *SkeletonGraph*: it has a reference to the driver of an algorithm graph and encapsulates tools for performing some graph operations. It can be seen as the root of the workflow process.

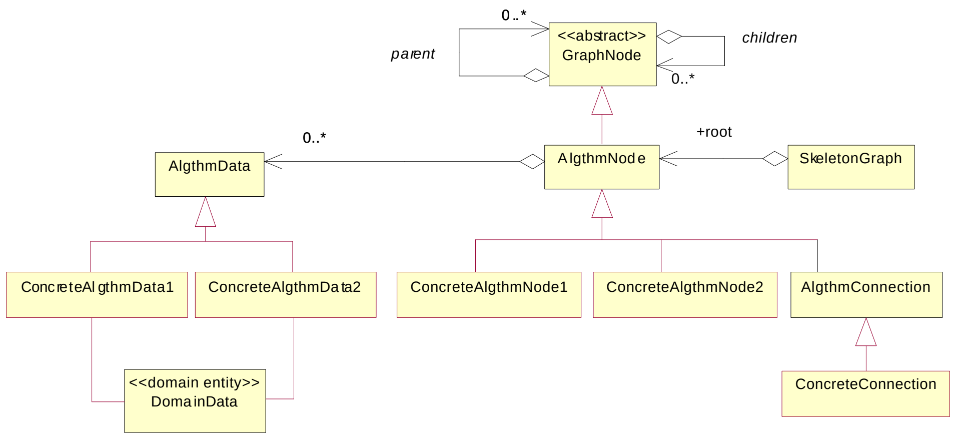


Figure 5-19 Participants of the GIG-pattern

- *AlgthmNode*: represent subroutines that compose the application (workflow tasks). It is used as a base class for all algorithm classes of the application.
- *ConcreteAlgthmNode* Implements a specific subroutine for a task. It invokes other subroutines which can be tasks (defined as its children) or other defined applications.
- *AlgthmData*: represents a data type to be used by an instance of an *AlgthmNode*. It is used as a base class for all algorithm data classes of the application.
- *ConcreteAlgthmData* Represents data from the application domain, which is used in *ConcreteAlgthmNode* classes.
- *DomainData*: represents the complete set of types related to the problem domain data.
- *AlgthmConnection*: this is an *AlgthmNode*, which references an algorithm subroutine that was not connected to the graph. This class responsibility is to fetch, and build (like a proxy [GHJ95]) the related algorithm and replaces itself with the fetched algorithm. In this way several software processes represented by *SkeletonGraph*s can be assembled producing a complex software system.

5.2.8 Collaborations

We can identify the following collaborations between GIG participants, see Figure 5-20:

- *GraphNode* encapsulates the responsibility of providing access to other *GraphNodes*, which are its children.

- *ConccreteAlgthmNode* executes the associated process (subroutine) with the help of other processes represented by its children, through calls inserted in its process code. It relies on *GraphNode* to have access to its children *AlgthmNodes*.

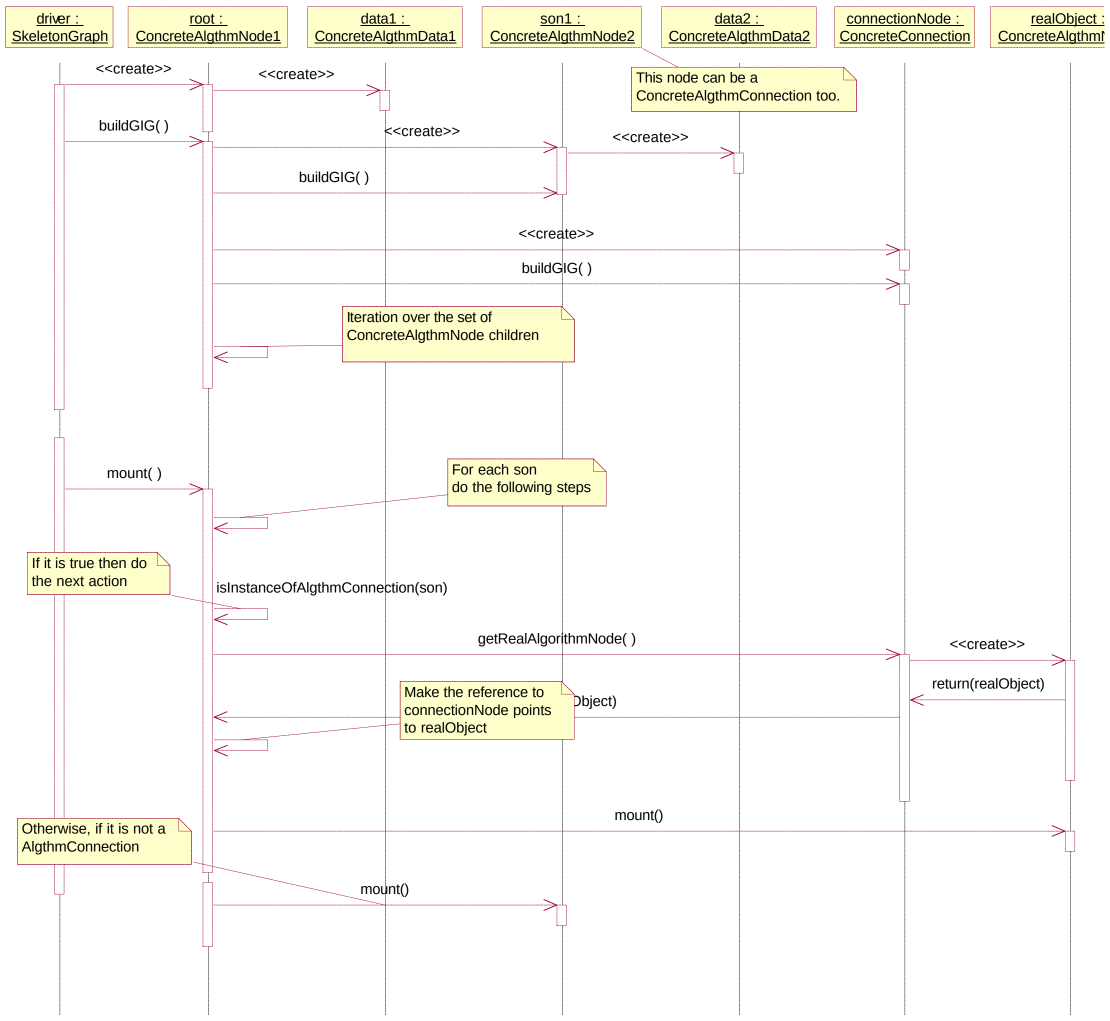

driver : SkeletonGraph
root : ConcreteAlgthmNode1
data1 : ConcreteAlgthmData1
son1 : ConcreteAlgthmNode2
data2 : ConcreteAlgthmData2
connectionNode : ConcreteConnection
realObject : ConcreteAlgthmN
This node can be a ConcreteAlgthmConnection too.
<<create>>
<<create>>
buildGIG( )
<<create>>
<<create>>
buildGIG( )
<<create>>
buildGIG( )
Iteration over the set of ConcreteAlgthmNode children
mount( )
For each son do the following steps
If it is true then do the next action
isInstanceOfAlgthmConnection(son)
getRealAlgorithmNode( )
<<create>>
return(realObject)
Make the reference to connectionNode points to realObject
Object)
mount()
Otherwise, if it is not a AlgthmConnection
mount()

Figure 5-20 Sequence diagram for GIG building

- *ConcreteAlgthmData* provides access to workflow data. The *AlgthmNode* communicates with Concrete*AlgthmData* objects to have access to its data.
- *The AlgthmConnection* provides the dynamic connection for *AlgthmNodes*. The way objects of this class interact with its *SkeletonGraph* or its parent *AlgthmNodes* depends on the implementation. The important thing is that it represents the point where a driver node of a software process/subroutine will

be plugged in. It also contains the necessary information about the new *AlgthmNode*.

Implementation

There are some implementation issues associated with the GIG participants, described previously, which need some extra explanation. Other important details about implementation are related to the design steps to be followed by the user when applying the GIG-pattern to a new application.

5.4.7 Implementation Issues

The *DomainData* is implemented by a set of subclasses of the *AlgthmData*. The subclasses of *AlgthmData* describe the specific domain treated in the problem. The *AlgthmData* and *AlgthmNode* objects must be materialised for the workflow they are serving. The materialization activities, of *AlgthmData* and *AlgthmNode* objects, can be delegated to object factories that are responsible for accessing the data repository and instantiating the objects. These object factories can have object pools to reuse objects, see section 5.4.11 (Related patterns) for details about the patterns that can be applied.

The *AlgthmNode* subclasses need to cast the *AlgthmData* objects, associated with each node, to the primitive type. As was shown before, in Figure 5-19, each AlgthmNode object must have a reference for all of its children and data. This reference can be hard coded in an AlgthmNode subclass, or in a file, or can be handled by another class, which has the responsibility to relate each AlgthmNode to its children. An example of such a class is the DataAlgthmServer use in the example (in section 5.4.9). In this case each AlgthmNode can ask the DataAlgthmServer for its children and data or the DataAlgthmServer can be active and responsible for building the GIG.

Design Steps

The following design steps describe which actions the user needs to perform to apply the GIG-pattern to a problem:

- Starting from an algorithm in natural language the process is first divided into different subroutines (algorithm nodes) ehich are organized in the form of a graph.
- The division of the algorithm into several subroutines induces a decomposition of the domain data in order to provide them with an appropriate distribution of access to the data. The result of this process gives the *AlgthmData* set.
- Each *AlgthmNode* places calls to its children nodes, which implement subprocesses that take part of the whole process. The logic is defined inside

each *AlgthmNode* subclass and it references the execution of a child algorithm, independently of the task of that child.

- Each *AlgthmNode* is related to a set of *AlgthmData*, which may be shared with other nodes.
- The driver of the whole process is identified.

5.4.8 Variants

(i) Use of the TypeObject pattern [YJ02] to enhance adaptability that produces independence between the software routines and its data components. This is important in situations where the same software component is to be used in different situations and with different pieces of data. The class diagram is similar to the one in Figure 5-21. With this extension, AlgthmType provides the AlgthmNode with the required functionality independently of AlgthmData. The relationship between AlgthmData and DataType can be made at run time. This extension does not affect interactions of AlgthmNode and AlgthmData with the other participants as already described.

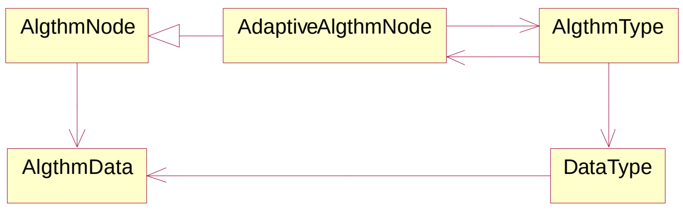


Figure 5-21 Class diagram for a variant of an AlgthmNode

(ii) Hierarchical levels of procedures can be defined to support software management. An application of this extension can be seen in section 5.4.9, where three levels of SkeletonGraphs were defined. For each level one may define specific functionalities for all their respective AlgthmNodes and AlgthmData. Furthermore, at the level of the functionalities of SkeletonGraphs, object specific tools can be defined. These extensions can be oriented for the applications being considered.

(iii) The pattern can be extended to deal with the definition/execution of processes running in a distributed environment. We will not go into further details because this is still under development.

5.4.9 Example

This example is related to the application of GIG in FEM simulators. As usual, it is observed that an algorithm defined for the solution of a problem by the FEM has repeated hierarchical structures. Therefore a framework considering hierarchical levels of processes was used, where each level may have several possibilities of algorithms, and can be easily described by a GIG graph. The whole hierarchy is represented making the connections between the different levels and generating a complete graph. Global, Block and Group Skeletons, and Phenomena procedures define those levels. These levels satisfy a number of requirements, such as: (i) to separate less reusable modules from reusable ones; (ii) to make more comprehensible the decomposition of the simulation data amongst the several processes; (iii) to make possible the dynamic re-configuration of the simulator through the replacement of reusable modules.

The global Skeleton articulates the time loop (if present), adapts iterations and defines processes involving the call of Block Skeletons. Block Skeletons may define different solution strategies for different Groups, thus, articulating Group processes. Group Skeletons articulate their phenomena procedures in very specific less reusable ways. It is at this level that solvers for algebraic systems are applied. Phenomena are the abstraction of the entities being simulated. All those skeletons can be implemented as objects from classes following the GIG pattern (see Figure 5-22). Therefore, the GIG will allow the interoperability of the different levels of computation (by automatically plugging the lower level skeletons into the higher ones).

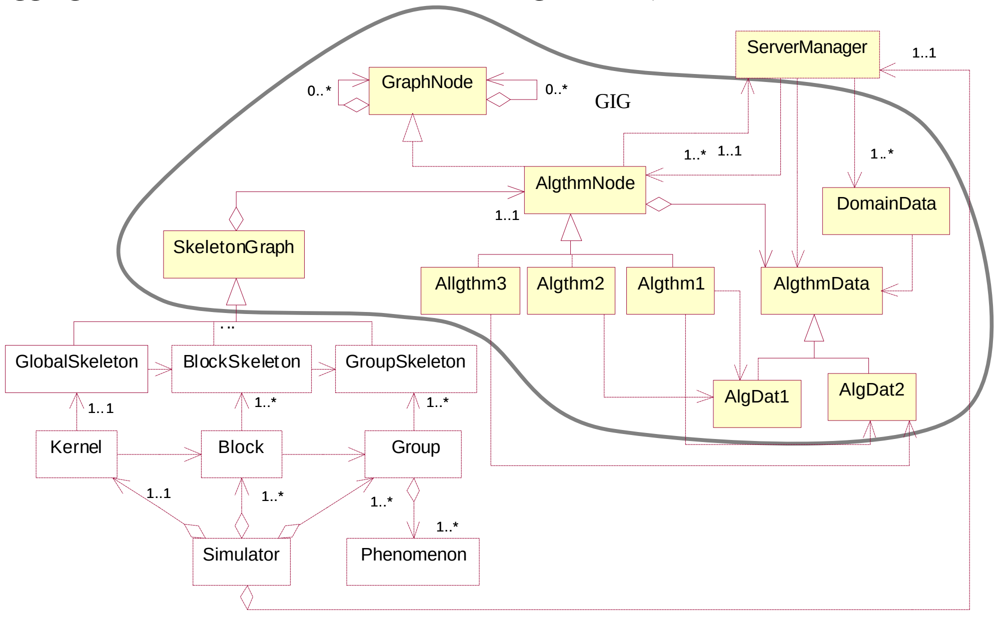


Figure 5-22 FEM Simulator and GIG classes

In the example described in what follows, we consider a FEM simulator specification. This kind of simulator is capable of solving, for example, problems involving transient phenomena, where the phenomena context includes linear temperature-dependent elasticity, rigid body motion and linear heat transfer (as Example 1 and 2 of Appendix A). In the present case n blocks are needed. The number of Groups depends on the phenomena types present in a specific simulation. The number and type of phenomena depends on the simulation being carried out as well. In the ith-Block Skeleton, Nig is its number of groups.

Figure 5-16 shows the Global Skeleton[1], while Figure 5-23 shows two Block Skeletons. Figure 5-24 presents the GIG direct acyclic graph to implement Global and Block Algorithm skeletons.

```
Is-Bi)Retrieve Initial State for Block i (see(I.0)):
Is-Bi.0)For r = 1 until Nig
 Is-Bi.0.0)Group r, compute phenomena initial states
It-Bi)Compute initial time step for Block i (see(I.I)):
It-Bi.0)For r = 1 until Nig
  It-Bi.0.0)Group r, compute Initial time step $\Delta r$
It-Bi.I)Set $\Delta ti = \min_{1 \le r \le Nig} \{\Delta r\}$
```

Figure 5-23 Block Algorithm Skeletons

As noted before, there should be AlgthmData objects, which contain the required problem and process data needed by each AlgthmNode object. A specialization of an AlgthmNode is an AlgthmConnection, which is defined whenever a lower level process is to be called up. Its AlgthmData object includes pieces of information needed for the identification of the lower level skeleton that will be plugged into the Algorithm Skeleton Graph. This identification concerns a driver AlgthmNode object (from another graph, integrating in this way the graphs presented in Figure 5-24), which will replace the related AlgthmConnection object.

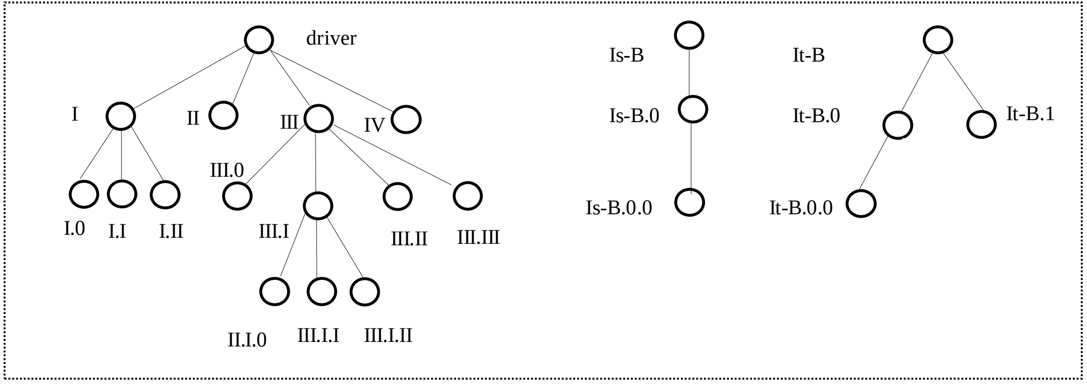


Figure 5-24 Global Alg.Skeleton graph **Block Alg.Skeletons graphs**

5.4.10 Consequences

Below we make some considerations about the forces related to this pattern.

We can observe **positive forces** for the use of the GIG-pattern:

- Easy translation from algorithmic language into computer processes. It supports an organisation at a graphic level, providing the distribution of code in a very

[1] The example used in the FEM-skeleton Pattern is also used in this subsection GIG-Pattern.

flexible way, not compelling a rigid division of code. To improve simplicity in process definition we try to: avoid unnecessary levels of details and to maintain similarity to the predefined algorithmic structure;

- Different users have evaluated this pattern with success, in applications with different levels of complexity [LSV03b];
- Support for different levels of granularity of the defined processes. It allows a flexible representation for a mixture of scales, since it does not restrict the levels of programming into which the code is defined, as opposed to [MAN01]. The workflow must be defined in terms of a set of node types that have already been coded in the programming language level;
- It can be applied to any domain solution, through the definition of specific domain data classes and algorithms, as can be seen from the pattern participants, in section 5.4.6;
- It allows the test of individual parts of the process independently, reducing the error occurrences in the coupling of processes;
- It allows the reuse of entire solutions, making changes in specific points. In GIG, it is easy to change parts of the graph, maintaining the others intact;
- It allows graph change (that is, the process change) at run-time. This is achieved through the GIG intrinsic dynamic structure, as was shown in section 5.4.6. Data and process can be defined at run-time, depending on GIG implementation, once a pattern can be easily extended to incorporate design patterns such as [YJ02], as presented previously.

Some **negative forces**, or restrictions, can also be identified:

- The pattern makes severe restrictions on the graph structure, requiring it to be an acyclic graph. The designer is not allowed to define neither recursive iterations nor loops outside of the node code;
- The GIG-pattern makes no explicit reference or imposition for the use of a specific set of process types, in contrast to [MAN01]. We can consider that this may cause a loss of workflow-refined control (that is, at instruction level). It is the programmer responsibility to define and manage this organization, if required by the application;
- The flow control is inside each node code. This can bring difficulties to some parts of process adaptation and control;
- Synchronization is not a GIG-pattern responsibility. A GIG does not define a specific structure to deal with process parallelism and processes synchronization. To allow the definition of processes parallelism, the programmer has to deal with extra complexity. A GIG-pattern requires unnecessary levels of repetition, that is, the replication of whole of process graph branches.

We may summarize saying that this pattern is **not very appropriate** for applications that are simple and do not require exchangeable processes, modularity or articulation of sub-routines. In addition, it is **inappropriate** for applications where there is a need for a high level of refinement in the program code, or if process synchronization and parallelism are required. In these cases, the Micro-workflow proposal is an alternative, as described in [MAN01]. However, through the use of the Micro-workflow alternative one of the worthwhile things you loose is simplicity and the level of granularity; the translation from algorithmic language is not such a direct mapping, losing in this way some levels of abstraction. The application of the GIG pattern to simplify applications can be more expensive then a simple solution. Conversely, it supports reuse, flexibility for new solutions, and domain independence.

5.4.11 Related patterns

The following patterns, can be used together with the GIG-pattern:

- Factory Method [GHJ95], which can be used to materialize objects for workflow management;
- Template Method [GHJ95], used to define skeletons of algorithms in *DataAlgthmServer* classes;
- Composite [GHJ95], used to implement the *AlgthmNode* class functionality in the framework.
- Proxy [GHJ95], used in the *AlgthmConnection* class;
- Strategy [GHJ95], used in AlgorithmNode and AlgorithmData classes
- Adaptive object-model patterns, such as TypeObject [YJ02], shown in the variants section.
- FEM-Simulator Skeleton [LSR02a] can benefit from the GIG approach.

5.4.12 Known uses

Many variations of numerical algorithms show the very same organizational structure, which was abstracted by the GIG-pattern. Examples include: mesh generation procedures, geometric reconstruction from planar slices and integration of geometric reconstruction procedures, etc.

Despite of being a generic solution that can be applied elsewhere, the users of this pattern have been scientists and engineers. The GIG-pattern has been applied with success in the development of different FEM simulator applications, and in a variety of other numerical methods in computational Mechanics. In Plexus we apply GIG as a general solution for the numerical methods and articulation strategies for solving groups of phenomena, see section (5.4.9).

5.5 Final Considerations

This chapter proposed patterns, which will give support to FEM simulators solutions domain: The Computational Phenomenon-Pattern, the FEM Simulator Skeleton Pattern and the GIG-Pattern.

The Computational Phenomenon Pattern represents an abstraction of the collection of commonalities found in the concepts and process for representing phenomena simulation through the FEM. It defines higher levels of abstraction and reusability.

The FEM Simulator Skeleton pattern supports the development of FEM simulators. It deals specifically with algorithm hierarchical modularisation for simulations based on the FEM. Hence, it is possible to separate complex procedures from simpler ones and strongly reusable software components from less re-usable ones. One immediate benefit is the enhancement of re-usability. It is worthwhile observing that each Simulator built, based on the proposed pattern, and should be able to solve new problems defined on completely different geometries and sets of phenomena.

The FEM-Simulator Skeleton pattern promoted:

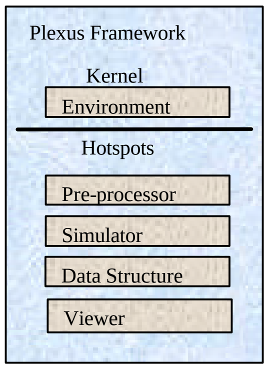
- Higher levels of abstraction for the main concepts of FEM Simulation Skeleton pattern modelling, reducing the complexity and improving the correctness of the systems (simulators) that will be developed.
- Higher level of hierarchical modularity for the system process organization, by the use of global skeletons, blocks and group skeletons.
- A solution, which may consider monolithic, coupled phenomena simulation.
- Reliability of the computer-generated predictions is improved by means of pre-defined strategies, numerical methods and templates.
- A higher level of maintainability, as the pattern separates different levels of computation and high reusability of the first two and last levels of computation are guaranteed.

In contrast, the GIG Pattern (Generic Interface Graph for Process Control) provided the flow process control for the defined simulator. Hence it became easier to translate from algorithmic language into computer processes, as well as achieving simpler process definition by avoiding unnecessary levels of details and maintaining similarity to predefined algorithmic structures. However, the pattern creates severe restrictions on the graph structure, requiring it to be an acyclic graph; the designer was not allowed to define neither recursive iterations nor loops out of the node code. Then again, it allowed the reuse of entire solutions, locating changes in specific points. It also opened up to support run time adaptivity processes.
These patterns address the following non-functional requirements:

- System flexibilization: (1) Support adaptability for changing numerical methods. Numerical algorithms were not hard coded, but treated as flexibilization points of the simulators abstraction. The algorithms used are customisable through configuration or extension of defined models; this occurs in the Computational Phenomena Pattern and also in the Skeleton Pattern. (2) The definition of an adaptive workflow management framework, where the designer workflow can be changed dynamically (Generic Interface Graph for Process Control, the GIG-pattern).

- The support for the definition of new simulation strategies is related to modular decomposition in parts of the problem, which are intrinsically related to the kind of global algorithm of the simulators. With proper modularisation, solution strategies can be changed without changing the basic structure. This is treated by the simulator model definition and controlled by the GIG-Pattern. The designer will deal with different kinds of simulators, depending on the global scenario it wants to use. The simulator will be dynamically adaptive.
- The reusability was considered in the reuse of: simulator models, simulation problem data, numerical solutions, and simulator specific data (phenomena, geometries, components, etc). The reusability requires a good conceptualisation of reality, which is supported by modelling pattern definition that is descriptions of abstractions about simulator concepts composition and functioning. Also the process classification, in different levels of computation (FEM-Skeleton Simulator), facilitates the process reuse.

This chapter also refers to Plexus Frameworks, that is, the Simulator Framework and the Computational Phenomena Framework. In spite of not being referred to here, we can abstract the whole Plexus system as a Framework that has more general hotspots Figure 5-25.



The Plexus Framework:
We can pre-view the building of a generic framework, which presents several flexible hotspots for the simulator development, such as the pre-processor, the simulator, the data-structure and the viewer machine. The architecture was defined in order to make them. This work, however was only interested in the modularisation of the involved components guaranteeing more independence between the major processes

Figure 5-25 Future Plans for the Plexus Framework

The Plexus Frameworks abstraction considers that, due to Plexus architectural modularisation, we can have hotspots such as Pre-processors, Simulators and Viewers that will be included (adapting according to simulation requirements). However this will be considered in a future work. We can also generalize some common features to also support a product line definition for the development of FEM simulators. The product line approach [KLD02] has been specially adapted for the development of product families, where multi-resolution modelling is explored.

The next chapter makes some final considerations about this work, showing future activities and suggestions for future work.

Chapter 6

Conclusion

This chapter summarizes the objectives and contributions of this work in the conceptualisation of a Simulation Environment, it describes some identified limitations, makes a comparison with other approaches, and proposes future activities.

6.1 Objectives of this Work

The world has seen many advances over the past three decades in modelling and simulation. However, methods of modelling and simulation are fragmented across disciplines making it difficult to reuse ideas from other disciplines and to work collaboratively in multidisciplinary teams [ZPK00].

The motivation of this work is to provide support to engineers and scientists for the modelling and control of simulators related to coupled multi-physics phenomena based on the FEM, taking advantage of the polymorphic nature of the method. Thus, a Simulation Environment, named Plexus, is proposed.

The expected results are the simplification of the definition of new solution strategies, support of model reuse, and the proposal of higher levels of abstraction related to the main concepts involved in the development of simulators based on the FEM. We help the user by providing abstraction mechanisms related to coupled phenomena, articulating different solution strategies for different phenomena groups; giving flexibility to define algorithms in several levels of the simulation. The use of data repositories based on a previously defined modelling patterns may significantly expand domain knowledge and its accessibility. This is inline with the idea that there should be a Numerical Analysis software repository that is operated by people with specific duties in data treatment and control, elected leaders, and public guidelines for what is deposited [RG00].

There are some related approaches that may provide contributions and improvements to FEM simulations. They define good practices and fundamentals but the existing contributions do not treat problems related to abstraction of numerical algorithms; easier change of numerical methods and strategies; satisfactory abstractions for couplings that can be defined independently of the actual implementation of the involved phenomena; abstractions for groups of phenomena, that are solved together; and abstractions of the relationship between geometry and phenomena.

In order to meet its objective, this work proposed an object-oriented architecture, which applies framework abstraction and patterns. These framework abstraction and patterns describe ideas and perspectives that have been observed and analysed during many daily studies that applied the FEM. Furthermore, the framework abstraction was used to represent domain specific conceptualisations, giving rise to a promising way of reusing simulator software designs and implementation. Reusing architectural structures is an advantage because the architecture is a pivotal part of any system and costly to construct.

The next section summarises how the work objectives have been addressed, and discusses their contributions. Some limitations are also presented and a comparison to other approach is made. Then, further work is suggested to overcome the method's limitations, improve it and expand its use. Next, some final remarks are given.

6.2 Contributions

The Plexus Simulation Environment conceptualisation focused on many objectives, which include: (i) to reduce simulator development complexity; (ii) to support flexibility in the use of FEM in multi-physics simulations; (iii) to decrease time spent by engineers in the design and production of simulators and (iv) to support simulation of coupled multi-physics phenomena.

The main contribution of this work is the conceptualisation of an environment, where reusable semi-complete applications for FEM simulators can be developed. These applications are frameworks, which can be specialized and customized to produce simulator applications. The proposed environment tries to apply relevant and standardized solutions - in the pattern sense - for FEM simulator domains.

This work elaborates a background description to help to understand FEM simulation of coupled multi-physics systems, detailing involved processes and concepts particular to the target project. Important issues related to the FEM simulation area were identified and existing works and studies were presented. The main proposed solutions, of this work, include:

i) Proper definition of requirements, through the identification of a flexible technique (Problem Frames) to describe the real world and also for specifying the involved requirements for the simulator to be developed;

ii) Definition of a generic architecture for a Plexus Environment, which will manage commonality across different simulators;

iii) Definition of different patterns for:
   - The computational phenomenon;
   - The simulator modelling, taking benefit of the FEM domain;
   - Control of the involved processes, guaranteeing solution independence simplicity and adaptability in execution time.

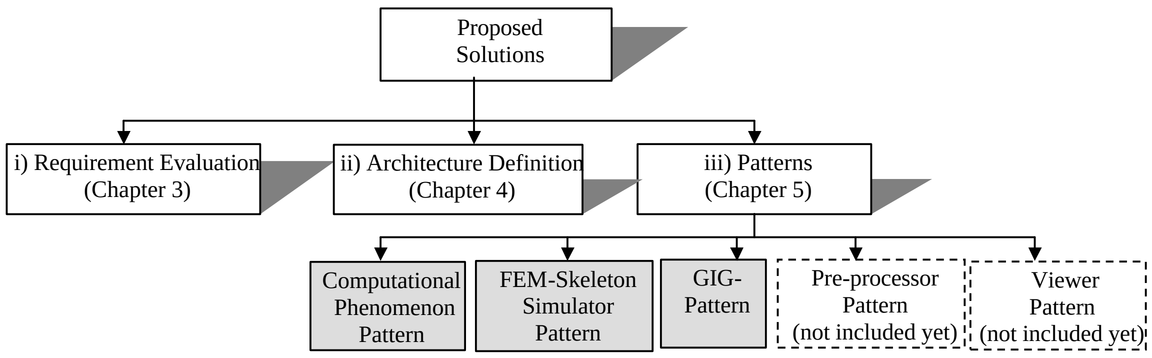

Proposed Solutions
i) Requirement Evaluation (Chapter 3)
ii) Architecture Definition (Chapter 4)
iii) Patterns (Chapter 5)
Computational Phenomenon Pattern
FEM-Skeleton Simulator Pattern
GIG-Pattern
Pre-processor Pattern (not included yet)
Viewer Pattern (not included yet)

Figure 6-1 Summary of the Proposed Solutions

Figure 6-1 gives an overview of the proposed solutions, and also references the definition of the Pre-processor Pattern and the Visualization Pattern, which are not an integral part of this document. However, each of them has references ([LSA02b] and [VA02] respectively) that will be used in future works for further developments.

In the sequence, we make a cross reference between the requirements and the proposed solutions addressed in this work to satisfy those requirements, highlighting our main contribution.

The **Domain Analysis** and **Requirement Evaluation,** in Chapter 3, presented a way to describe FEM simulators with the Problem Frames software engineering technique - one of the most respected software engineering approaches for requirements analysis and problem domain specification - improving the description of our specific domain. That chapter detailed the problem domain, analysis requirements and made the problem decomposition. We evaluated the appropriateness of the technique, discussed its power of expressiveness and limitations, suggesting what could be improved. The conclusions of our analysis about Problem Frames are relevant and might be helpful for other domains of knowledge, which have similar characteristics, such as a strong multidisciplinary nature, which causes difficulties in the elaboration of abstractions involving different knowledge domains. The approach addresses the goals related to the: (i) improvement of domain comprehension, making the involved concepts clearer and formalized, allowing them to be more correctly implemented, modified or reused; (ii) simplification of requirements specification, through the identification of the systems major requirements, which will decrease the number of errors or the lack of documentation, and simplifying the future processes of requirements analysis.

The **Architectural Definition** from Chapter 4, proposed a specific way to deal with a single solution for the development of coupled phenomena simulators based on the FEM. This architecture, defines a structure for supporting the development (Creation, Configuration, Setup, Load and Use) of simulators for coupled phenomena problems, based on FEM solutions. The description of a well-defined architecture was crucial to guarantee the quality, reusability, and decrease costs and complexity in the definition of simulation models. This also simplified the definition of new solution strategies. We can argue that the described architecture gave a high level of abstraction; promoted *reuse* of the involved components; supported framework *construction*; helped to identify the need to support a distributed and cooperative environment; enabled framework *analysis* in agreement with quality attributes. The conceptualisation of the Plexus environment architecture is considered a relevant contribution, as it is a mechanism to obtain the required reuse of software components in FEM simulator systems [LS03].

Frameworks and **Pattern definitions** are provided in Chapter 5. The FEM **simulator framework definition** was very appropriate to our domain where similar applications are built several times from scratch. Special attention was also directed to the **computational phenomenon** abstraction, which was also considered as a framework. The defined frameworks gave support to ways of reducing the costs and improving the quality of simulation software. In contrast to an earlier object-oriented reuse technique, based on class libraries, our work defined frameworks targeted to particular simulation process units (such as simulators, workflow management, visualization) and application domain (coupled multi-physics phenomena simulation based on the FEM). The **Pattern Definition** includes patterns related to FEM simulator solution domains (phenomena modelling, process reuse, simulator modeling and process control flow). Researchers of patterns have shown that patterns are effective tools for reuse. Pattern definition also promotes the achievement of software quality. Our work addresses this novel area of FEM simulators where we had great gains in process reuse. Engineers require specific and better solutions, which could explore in detail the expressiveness and reusable capacity of their specific domain. Definition of some abstraction considering the main concepts, such as simulators and computational phenomena, were a concern, which was explored by framework abstractions.

Examples of FEM simulator problems illustrating coupled multi-physic phenomena are described and presented in Appendix A. One of them, the Example 1, is used as a case study during the description of above mentioned patterns.

A **Prototype** of the Plexus Simulation Environment is under development in LINUX platform, using C++ language. A pre-defined knowledge base, including FEM simulators meta-data, was also defined. The database use was encouraged by the need of more powerful features that reduce time and effort. The use of a DBMS (in our case the Postgres Database Management System) makes the persistent data management easier, guaranteeing the control, sharing and reuse of large amount of simulation data and concurrence control, while supporting data integrity in case of system's failure.

The **current implemented parts** correspond to: part of the knowledge base management implementation and the GIG-pattern implementation. Appendix C shows some of the prototype interface windows. In future works, our aim is to finish the whole environment implementation.

6.3 Limitations

Some limitations can be identified. Firstly, considering the applied technique, Problem Frames, for requirements analysis, we must point out that several concepts require experience, which could take a long time to acquire. This comes out of the fact that this technique is neither trivial nor intuitive. However, many existing concepts make it possible to describe the problem and the knowledge domain in a powerful way, as well as the requirements specification. Another limitation is that there is no complete meta-model definition, which represent the whole proposed model. In addition, despite the supported features that the designer can use, some reasonable familiarity with the environment and existing framework is required. This demands extra effort before it starts to meet the provided benefits. Also, the designer has the responsibility to organize its code and data in a way that will explore the defined advantages of the Plexus Simulation Environment. An example is the FEM Simulator Skeleton Pattern, where some limitations can be identified: the model builders require special training, that is, the designer must understand the proposed abstractions; designers will only achieve higher levels of reusability if they know how to articulate their strategies and problems; the simulator performance can decrease due to extra imposed levels of abstraction. Other limitations, such as not treating simulation distribution, come from our objective to simplify our first approach to the proposed environment. Most of the Plexus validation was done through Brainstorming Meeting [], where during several meetings specialists gave opinion and evaluation of the proposed solutions. The submission and acceptance of papers related to Plexus also can be considered a validation aspect. However it is fundamental to make a more rigorous and complete validation of the environment applying measurement technologies and experimental methods to the proposed software environment [PKL01, KIT02].

6.4 Comparison between Plexus and other approaches

As we can see from chapter 2 and Appendix B there are several approaches for scientific simulation development from which FEM simulators can be built. We can classify them as:

a) Pre-built Libraries such as DIFFPACK [LAN97] and PZ [DP99, DP04], which are natural complements of general-purpose languages, which allow saving in time and the guaranty of main advantages (e.g. use of already developed complex and proved code).

b) General purpose programming environments (e.g. MATLAB [MAT04], SCIRUN [PWC97]), where there is support for the simulator and simulation development however they do not explore the FEM characteristics in a higher level (such as at the level of the simulator, for example). So the user has the possibility to build its own solution, not being subjected to environment restrictions; however it requires more knowledge and hard work to develop complex simulators.

c) Domain specific rigid environments (e.g. ANSYS [ANS04]) where, in spite of the high sophistication level to solve specific problems, there is neither support nor flexibility to change the way the problems are solved.

d) Domain specific modelling environments where despite of the support of pre-defined solutions the user has also the opportunity to develop new solutions from scratch, however with an underlying guide and support to do so (e.g FEMLAB [FEL04] and Plexus).

Since FEMLAB is the most approximate approach considering our proposed solution, next we will make a comparison between FEMLAB and Plexus environment.

Our analysis considers some specific points, described next (see also Table 6-1):

A) Levels of proposed abstraction

Despite of the importance of the separation of concerns in a specific field, in order to reuse knowledge and provide experience and data reuse, there are few approaches, which describe good levels of abstractions. FEMLAB is a case of a FEM approach, which includes some specific abstractions, which improve the level of the supplied flexibility and system power. It supports multi-physic modelling and simulation for coupling of multi-physic phenomena. You can build complex models by combining several of the package’s integrated ready-to-use applications modes or using equations–based modelling. However, it does not support some relevant abstractions, as the ones used in Plexus for the definition of solution strategies and for the automatic construction of simulators (presented in Chapter 5).

B) Support for an integral piece of software for coupled phenomena simulations

There is a great need for integral solutions, which give support for the development of simulations involving several coupled phenomena, and also the need for supporting an unlimited multi-physics combination of coupled phenomena, as mentioned in chapter 2. FEMLAB provides a multi-physic modelling and simulation environment for coupling of multi-physic phenomena. You can build complex models by combining several of the package's integrated ready-to-use applications modes or using equations–based modelling. Plexus gives more flexibility, because, in spite of supporting an unlimited combination of multi-physics phenomena, it also allows different meshes for phenomena defined in the same geometry. Also, Plexus allows the user to develop the solution strategies, which define the way and the order in which each phenomenon is used during the simulation. Femlab does not support this.

C) Flexibility in the implementation of different numerical methods for the same task

There is a great need for implementing flexibility for changing auxiliary numerical methods, solution methods, error estimation, adaptation methods, shape functions and viewers. We can consider for example the support for the definition of a user required mesh generator. In FEMLAB we can choose from different pre-defined kinds of mesh generators. However in Plexus we have the possibility to incorporate an unlimited number of methods for this task.

D) Flexibility in the implementation of different simulators strategies

To our knowledge there is no FEM-specific development environment, which allows and gives support (through abstractions) for the changing of the simulator solution schemes. One of the main driving forces for Plexus development was to provide that kind of support. So, high levels of abstractions were developed targeting that requirement. In FEMLAB we can identify that at some level the user can make some articulations in the way the coupled phenomena are connected. However, Plexus goes further when it provides abstractions, which orient the definition of new simulators from scratch. After a simulator is built, those abstractions also allow for the simulator to be reconfigured. Thus, Plexus provides different ways of articulating blocks and groups of procedures.

E) Support for workflow

The use of workflow technology helps the development of more flexible and versatile computation strategies, as described in chapter 5 (GIG-pattern). The workflow control and use provide expressiveness and adaptability through simplification of the system being developed. Plexus integrates the workflow concept in its solution at the user level, but FEMLAB does not.

F) Reuse of experience (reusability)

The reuse of experience can be measured through the support of a repository, the reuse of simulator models (strategies), reuse of problems definition (geometries, phenomena, etc), and so on. We can notice that Plexus scope of reusability is larger than FEMLAB, since it stores information considering the knowledge basic level (semi-complete data, like phenomenon, numerical methods, etc.), simulator level and problem level.

Table 6-1 Comparison between Plexus and other approaches

Criteria	FemLab	Plexus
A) Levels of abstractions	++	+++
B) Flexibility for solution methods	+	+++
C) Flexibility in the simulator definition	+	+++
D) Support for phenomena coupling	+++	+++
E) Support for workflow at user level	-	+++
F) Reusability	++	+++

So we can notice that Plexus main contributions include aspects related to:

- Single Software Solution for coupled multi-physic simulation based on FEM.
- Flexibility in the implementation of different simulators strategies.
- Make the simulators and problem development easier and faster than before, through the use of proposed abstractions.
- Reuse of experience.
- Support for the definition of new simulation strategies.
- Support for the implementation of different numerical methods for the same task.

6.5 Final Remarks

There are several alternatives for FEM simulator development. The solutions presented in this work undertake some of the aspects that have been neglected, such as the need for specific abstractions, models and process domain definition of the specific area. The exploration of domain specific solutions makes it possible to achieve better results, which frequently are more difficult to obtain by general solutions.

The development of FEM simulators is a key activity in many engineering areas. Special attention must be continuously given to abstraction definition, procedure reuse and simulator modelling. Another relevant point is considering the involved designers. Simulator applications are becoming more and more complex and more reliable systems are needed. To find domain specific solutions is a priority. However, in order to achieve this, the researcher should be aware of the relevant problems, and should know what they really need. In this work we try to improve a little bit on domain comprehension (indicative and optative) moods, as referenced by [JAC01], specific solutions in the FEM simulation domain were suggested. The development of a reusable system takes advantages of abstractions based on application examples applied in many and diverse situations by the area specialists. The continuity of this task is a challenge, which can bring many gains in different and important areas of human life, where a FEM application is an effective solution.

6.6 Future Work

Much more must be done to improve our work, and can be considered as future activities, outside of the scope of this thesis. Some possible future extensions are listed bellow:

- To make the simulation environment more automatic, since there are lots of standard situations and states, which are neither assisted nor guaranteed, which can be improved through the inclusion of higher levels of expertise;
- The integral environment detailing and implementation;
- To include a tool to monitor the simulation process;
- Explore workflow engine features;
- Explore and provide parallelism;
- To ensure consistency, security and performance evaluation;
- To make an extension of the defined architecture to comply with HLA architecture standards for partitioning a simulation in different simulations that can be distributed across multiple computers or processors;
- Simulations frequently require large volumes of data, which can be partitioned and processed by many processors in a distributed memory environment. So, it is important to analyse the scalability with respect to the number of processors and also to analyse the distribution of data [LF99, LCF00];
- Despite gains already identified, it is important also to quantify them in order to demonstrate the Plexus Simulation Environment power;
- Observing framework leverage domain knowledge to support the development of product lines, that is, families of related applications [FAY97]. In future works, we will make some considerations about product lines in simulation environments based on the FEM, that is, a product line architecture designed to support the variation needed by the products, and so making it re-configurable;
- Exploration of the proposed Plexus framework in order to apply it to different simulation methods, like the finite volume method [VM95]. The main considerations regard the pre-processing methods and the level of the computational phenomenon.

Bibliography

[AB00] Appleton, B., “Patterns and Software: Essential Concepts and Terminology”, 2000. Available at http://www.bradapp.net, accessed on 10/01/2004.

[ALX77] Alexander, C.; Ishikawa S.; Silverstein, M.; Jacobson, M.; Fikdahl-King; Angel S., “*A Pattern Language*”. England: Oxford University Press, 1977.

[ALX79] Alexander, C., “*Small Best Practices Patterns*”. England: Oxford University Press, 1979.

[ANS04] ANSYS Incorporation, *“ANSYS”*. Available at www.ansys.com, accessed on 02/02/2004.

[AVS02] AVS., “*Advanced Visualization System - 2002*”. Available at: http://www.avs.com, accessed on: 3/01/2002.

[BAJ98] Banks, J., “*Handbook of Simulation - Principles, Methodology, Applications and Practice*”. John Wiley & Sons, Inc. Georgia Institute of Technology, Atlanta, Georgia, 1998.

[BBG97] Bateman, R.; Bowden, R.; Gogg, T.; Harrell, C.; Mott, J., “*Improvement Using Simulation*”. 5th edition, PROMODEL Corporation, Oem, Utah, 1997.

[BCK98] Bass, L.; Clements, P.; Kazman, R., “*Software Architecture in Practice”*. Addison-Wesley, 1998.

[BM002] Bosch, J. et al. *“Software Architecture System Design, Development and Maintenance”*. Edited by J. Bosch, M. Gentleman, C. Hofmeister, and J. Kuusela; Kluwer, Academic Publishers, 2002.

[BUB93] Bubenko, J., “Extending the Scope of Information Modelling DAISD, pp. 73-97, 1993.

[BUB95] Bubenko, J.; Song, W., Johnesson, P., *“Challenges in Requirements Engineering”. Proceedings of RE’95*, pp. 160-163, 1995.

[BUS96] Buschman, F.; Meunier, R.; Rohnert, H.; Sommerland, P.; Stal M., “*Pattern Oriented Software Architecture: A system of Patterns*”. New York: Wiley, 1996.

[CFM02] Casati F.; Fugini, M.; Mirbel, I.; Pernici B., *“WIRES: A methodology for developing Workflow Applications”*. Requirements Engineering’02, 7(2), pp. 73-106. Edited by P. Loucopoulos and J. Mylopoulos, 2002.

[CLR01] Cormen, T.; Leiserson, C.; Rivest, R., et all. *“Introduction to Algorithms”*. Second edition, MIT Press, 2001.

[CNL01] Clements, P.; Northrop, L., *“Software Product Lines: Practices and Patterns”*. Addison –Wesley, 2001.

[COD92] Coad, P., “*Object Oriented Patterns*”. Communications of the ACM, 35(9), pp. 152-159, 1992.

[COM00] Committee on Theoretical and Applied Mechanics, “*Research Directions in Computation al Mechanics*”. A report of the United States Association for Computational Mechanics, 2000. Available at http://www.usacm.org/org_cm.htm, accessed on 10/01/2004.

[COP01] Coplien, J., “*Foundation of Pattern Concepts and Pattern Writing*”. Bell Labs/ USA and University of Manchester / UK., 2001.

[COR95] Cook, R., “*Finite Element Modelling for Stress Analysis*”. By John Wiley and Sons, Inc., 1995.

[CR96] Crain, R., “*Simulation using GPSS/H*”. In Proceedings of the 1996 Winter Simulation Conference, J. Charnes, D. Morrice, D. Brunner, and J. Swain (eds.), Association for Computing Machinery, pp.31-38, New York, 1996.

[DK02] Dale, K., “*The Value of Quality Simulation Conceptual Model*”. The Johns Hopkins University Applied Physics Laboratory, Modelling Simulation Magazine, 1(1), a publication of the Society for Modelling and Simulation International, 2002.

[DLF93] Dardenne, A.; Lamsweerde, A.; Fickas, S., “*Goal-directed Requirements Acquisition*”. Science of Computer Programming, vol. 20, pp. 3-50, North Holland, 1993.

[DP04] Devloo, P.R., “*PZ - Environment for developing finite element software*”. Available at www.fec.unicamp.br/~phil, accessed on 29/01/2004.

[DP99] Devloo, P.R., “*Object Oriented Tools for Scientific Computing*”. Available at www.fec.unicamp.br/~phil, accessed on 29/01/2004.

[FAA00] Filho, A. “*Elementos Finitos – A Base da Tecnologia CAE*”, Editora Erica, 2000.

[FAY00] Fayad, M.; Douglas, S.; Johnson, R., “*Domain Specific Application Frameworks: Frameworks Experience by Industry*”. Wiley Computer Publishing, 2000.

[FAY97] Fayad, M.; Schmidt, D., “*Object Oriented Application Frameworks*”. Communication of the ACM, 1997.

[FAY99] Fayad, M.; Douglas, S.; Johnson, R., “*Building Application Frameworks: Object-Oriented Foundations of Framework Design*”. Wiley Computer Publishing, 1999.

[FEL04] Finite Element Modeling Laboratory, “*FEMLAB*”, *Available at* http://www.comsol.com/, accessed on 09/02/2004.

[FJL001] Fiorini, S.; Leite, J.; Lucena, C., Proceedings of CaiSE 2001, LNCS, pp. 284-298, 2001.

[FP01] Filho, P., “*Introdução a Modelagem e Simulação de Sistemas*”, Visual Books ed., Santa Catarina, 2001.

[GHJ95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides J., “*Design Patterns: Elements of Reusable Object- Oriented Software*”. Addison-Wesley, Massachesetts, 1995.

[GKP99] Garlan, D.; Kompanek, A.; Pinto, P., “*Reconciling the needs of architectural description with object-modelling notations*”. Technical report, Carnegie Mellon University, 1999.

[GR02] Grcar, J., “*Regarding Numerical Software*”. Personal Communication, posted at Lawrence Berkeley National Laboratory, Mail Stop 50A-1148, One Cyclotron Road, Berkley, CA 94720 (jfgrcar@lbl.gov), 2002. NA Digest Sunday, March 31, 2002, 02(13), NA-net. http://www.netlib.org/na-digest-html/02/v02n13.html#5.

[GSP00] Garlan, D.; Sousa, P., “*Documenting Software Architectures: Recommendations for Industrial Practice*”. CMU-CS-00-169, 2000.

[GV95] Gomes, J.; Velho, L., “*Computação Gráfica: Imagem*”. Sociedade Brasileira de Matemática, 1995.

[HNS99] Hofmeister, C.; Nord, R.; Soni, D., “*Describing Software Architecture with UML*”. In proceedings of the First Working IFIP Conference on Software Architecture (WICSA1), San Antonio, TX, February, 1999.

[HR99] Hilliard, R., “*Building Blocks for Extensibility in the UML: Response to UML 2.0 Request for Information*”. Integrated Systems and Internet Solutions Inc, Concord, Massachusetts, 1999.

[IBM02] IBM, “*Open Data Explorer*”. Available at: http://www.opendx.org, accessed on 3/01/2002.

[JAC01] Jackson, M., “*Problem Frames: Analysing and Structuring Software Development Problems*”. Addison-Wesley, 2001.

[JAC02] Jackson, M. et al., “*Relating Software Requirements and Architectures using Problem Frames*”. International Conference on Requirements Engineering (RE'02)**,** Essen, Germany, 2002.

[JAC95] Jackson, M., “*Software Requirements & Specifications: A lexicon of Practice, Principles and Prejudice*”. ACM Press, 1995.

[JAC96] Jackson, M., “*Four Dark Corners of Requirements Engineering*”. ACM Transactions on Software Engineering and Methodology, 6(1) pp.1-30, 1996.

[JAC97] Jackson, M., “*The Meaning of Requirements*”. Annals of Software Engineering, 1997.

[JF88] Johnson, R.; Foote, B., “*Designing Reusable Classes*”. Journal of Object Oriented Programming 2(1), pp.22-35, Jan-Feb, 1988.

[KG04] Kienbaum, G., “*Simulação de Sistemas*”. Instituto Nacional de Pesquisas Espaciais Laboratório Associado de Computação e Matemática Aplicada Simulação de Sistemas. Available at http://www.lac.inpe.br/~germano/SIMSIS/cap259/curso. htm, accessed on 10/01/2004.

[KHR04] Khoral Research, Inc., “KHOROS Scientific Environment”. Available at: http://www.tnt.uni-hannover.de/js/soft/imgproc/khoros/khoros2/ #Solutions, Accessible on 15/01/2004.

[KIT02] Kitchenham, B., “The question of scale economies in software - why cannot researchers agree?”. Information and Software Technology 44, 2002, pp 13-24.

[KL97] Kalasky, D.; Levasseur G., “*Using Simple++ improved modelling efficiencies and extending model life cycles*”. In Proceedings of the 1996 Winter Simulation Conference, S. Andradottir, K.J. Healy, D. H. Withers, and B. Nelson (eds.), Association for Computing Machinery, pp. 611-618, New York, 1997.

[KLD02] Kyo, K.; Lee, J.; Donohoe, P., “*Feature-Oriented Product Line Engineering*”. IEEE Software, 19(4) pp.58-65, 2002.

[KOR97] Kortright., “*Modelling and Simulation with UML and Java*”. Nicholls State Univ. LA, 1997.

[KWD99] Kuhl, F.; Weatherly, R.; Dahmann, J., “*Creating Computer Simulation Systems: An Introduction to High Level Architecture*”. Prentice Hall PTR, 1999.

[LAM01] Lamsweerde, A., “*Goal-Oriented Requirements Engineering: A Guided Tour*”. Appeared in Proceedings RE’01, 5th IEEE International Symposium on Requirements Engineering, pp. 249-263, Toronto, 2001.

[LAN01] Langtangen et al., “*Finite Element Pre-processors in Diffpack*”, Numerical Objects Report Series (Report 1999-01), Oslo, Norway, 2001.

[LAN97] Langtangen et al., “*Increasing the Efficiency and Reliability of Software Development for System PDEs*”. Modern Software Tools for Scientific Computing. pp. 247-268, in Erlend Arge, Are Magnus Bruaset, and Hans Petter Langtangen, eds., Birkhauser Boston, Cambridge, MA,1997.

[LCF00] Lencastre, M.; Paiva, M.; Castro, J., Fonseca, D. ,*“O Uso da Ferramenta NFR no Projeto de Banco de Dados Distribuído”*. III Workshop Iberoamericano on Requirements Engineering, R. Janeiro, Brazil, 2000.

[LCS03] Lencastre, M.; Castro, J.; Santos, F.; Araújo, J., “*Problem Frames Application On Finite Element Method Simulators*”. 6th Iberoamerican Workshop on Requirements Engineering and SW Environments (IDEAS'2003), pp. 274-279. Assuncion, Paraguay, 2003.

[LF98] Lencastre, M.; Fonseca, D., *“Uma Ferramenta para Projeto de Banco de Dados Distribuído”*. XXIV Conferência Latino Americana de Informática, pp.1081-1092, Equador, 1998.

[LF99] Lencastre, M.; Fonseca, D. ,*“ ProjetoOO - Ferramenta para Apoio ao Projeto de Banco de Dados Distribuídos”*. Logos Tempo e Ciência, Revista do Instituto Luterano Manaus, Manaus, 1(3), pp. 55-64, Brazil, 1999.

[LJ99] Leite, J., *“Are Domains Really Cost Effective?”*. Workshop on Institutionalizing Software Reuse, WISR'9, The University of Texas at Austin, 1999.

[LS03] Lencastre, M.; Santos, F., “An Approach for FEM Simulator Development”; Journal of Computational Methods in Sciences and Engineering (JCMSE), 2003.

[LSA01] Lencastre, M.; Santos, F.; Almeida, I., “*Data and Process Management in a FEM Simulation Environment for Coupled Multi-physics Phenomena*”. Proceedings of the 5th International Symposium on Computer Methods in Biomechanics and Biomedical Engineering, Rome, 2001.

[LSA02b] Lencastre, M.; Santos, F.; Araújo, J., *“A Process Model for FEM Simulation Support Development*. Proceedings of the Summer Computer Simulation Conference (SCSC 02), San Diego, California, 2002.

[LSR02a] Lencastre, M.; Santos, F.; Rodrigues, I., “*FEM Simulator based on Skeletons for Coupled Phenomena*”. Proceedings of the 2nd Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP'2002 Conference), pp.35-48, Brazil, 2002.

[LSR02b] Lencastre, M.; Santos, F.; Rodrigues I., “*FEM Simulation Environment for Coupled Multi-physics Phenomena*”, - Simulation and Planning In High Autonomy Systems, AIS2002; Theme: Towards Component-Based Modelling and Simulation. pp. 259-266. A publication of the Society for Modelling and Simulation International, Portugal, 2002.

[LSV03a] Lencastre, M.; Santos F.; Vieira, M., “*Workflow for Simulators based on Finite Element Method*”. Proceedings of the International Conference on Computational Science (ICCS 2003), Melbourne, Australia and Saint Petersburg, Russia. 1(2), pp. 555-565, Springer Verlag, 2003.

[LSV03b] Lencastre, M.; Santos, F.; Vieira, M., “*GIG-Pattern*”. Proceedings of the 3rd Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP'2003), pp. 293-307, Pernambuco, Brazil, 2003.

[MAN01] Manolescu, D., “*Micro-Workflow: A Workflow Architecture Supporting Compositional Object Oriented Software Development”*. Ph.D, Depart. of Computer Science University of Illinois at Urbana-Champaign, 2001.

[MAT04] MathWorks, Inc. *“MATLAB”*. Available at www.Mathworks.com, accessed on 02/02/2004.

[ME99] Medvidovic, N.; Egyed A., “*Extending Architectural Representation in UML with View Integration*”. Proceedings of the 2nd International Conference on the Unified Modelling Language (UML), pp. 2-16, Fort Collins, 1999.

[MM97] Markt, P.; Mayer, M., “*Witness Simulation Software: a flexible suite of Simulation Tools*”. In Proceedings of the 1997 Winter Simulation Conference, S. Andradottir, K. J. Healy D. H. Withers, and B. L. Nelson (eds.), Association for Computing Machinery, pp. 711-717, New York, 1997.

[MRC02] Martin, R. et al., “*Agile Software Development: principles, patterns, and practices”*. Published by Alan Apt Series, Prentice Hall, 2002.

[MY00] Mylopoulos, J.; Yu, E; Chung, L., “*Non-Functional Requirements in Software Engineering”*. Kluwer Academic Publishers, Boston/Dordresh/London, 2000.

[NCA01] Nelson, M.; Cowan, D.; Alencar, P., "*Geographic Problem Frames*". Poster on Proceedings of the 5th IEEE International Symposium on Requirements Engineering, pp. 306–307, 2001.

[NEI84] Neighbors, J., "*The Draco Approach to Constructing Software form Reusable Components*", IEEE Transactions on Software Engineering, 10(5), pp. 564-573, 1984.

[NG99] Nordstrom, G., "*Meta-modelling – Rapid Design and Evolution of Domain Specific Modelling Environments*". Dissertation for the degree of Doctor of Philosophy in Electrical Engineering of the Faculty of the Graduate School of Vanderbilt University Nashville, Tennessee, 1999.

[OK92] Oren, T.; King, D., "*Requirements for a Repository Based Simulation Environment*". Proceedings of the Winter Simulation Conference, edited by Swain, G. et al., 1992.

[OMG02] OMG: Unified Modeling Language 2.0. Initial submission to OMG RFP ad/00-09-01 (UML 2.0 Infrastructure RFP) and ad/00-09-02 (UML 2.0 Superstructure RFP).: Proposal version 0.63 (draft), 2002. Available at http://www.omg.org/

[OMG03] OMG., "*Unified Modelling Language Specification Version 1.5*", 2003. Available at: http://www.uml.org/, accessed on 10/01/2004.

[PJ70] Palme, J., "*SIMULA 67 – An advanced programming and simulation language*", Technical Report, Swedish Res. Inst of National Defence, 1970.

[PKL01] Pickard, L.; Kitchenham, B.; Linkman, S., "Using simulated data sets to compare data analysis techniques used for software cost modelling". IEE Proceedings on Software Engineering, 148 (6), pp165-148, 2001.

[PR96] Pree, W., "*Framework Patterns*". New York: SIGS Books, 1996.

[PSS90] Pedgen, C.; Shannon, R.; Sadowski, R., "*Introduction to Simulation using SIMAN*". McGraw-Hill, 2th ed., New York, 1990.

[PWC97] Parker, S.; Weinstein, D.; Christopher, J., "*The SCIRUN Computational Steering Software System*". In Erlend Arge, Are Magnus Bruaset, and Hans Petter Langtangen (eds.), Model Software Tools for Scientific Computing, Birkhauser Boston, Cambridge, MA, 1997.

[RD98] Roberts, C.; Dessouky, Y., "*An Overview of Object-Oriented Simulation*", IEE, Simulation: Transactions of The Society for Modelling and Simulation International, 70(6), pp. 359-368, 1998.

[REC93] Russell, E.C., "*SIMSCRIPT II.5 and SIMGRAPHICS Tutorial*". In Proceedings of the 1993 Winter Simulation Conference, G.W.Evans, M. Mollaghasemi, C. Russell, and W. E. Biles (eds.), Association for Computing Machinery, pp.223-227, New York, 1993.

[RG00] Ready, J.; Gartling, D., "*The Finite Element Method in Heat Transfer and Fluid Dynamics*", Second Edition, Lewis Publishers Inc. Salesrank, 2000.

[RJ96] Roberts, D.; Johnson, R., *"Evolving Frameworks: A Pattern Language for Developing Object Oriented Frameworks"*, University of Illions. Presented at PLoP '96, 1996.

[RM96] Rucki, Miller., *"An Algorithm Framework for Flexible Finite Element-based Modelling"*. Computer Methods Application Mechanical. Eng. 136, pp. 363-384, 1996.

[RN02] René, B.; Neves, S., *"Component-Based Software Development"*. MSc in Informatics Engineering, FCT, Universidade Nova de Lisboa, Portugal, 2002.

[RN95] Russel, S.; Norving P., *"A Modern Approach"*. Prentice Hall Series in Artificial Intelligence, 1995.

[RY01] Revault, N.; Yoder, J., *"Adaptive Object-Models and Meta-modelling Techniques"*. Workshop Results; ECOOP 2001, Hungary, Workshop Reader; Lecture Notes in Computer Science; Springer Verlag, 2001.

[SC03] Silva, C., *"Detalhando o Projeto Arquitetural no Desenvolvimento de Software Orientado a Agentes: O Caso Tropos"*. MSc Dissertation, CIN-UFPE-Brazil, 2003.

[SCS02] Society for Modelling and Simulation International, *"The value of a Quality Simulation Conceptual Model"*. Modelling Simulation, 1(1), 2002.

[SG96] Shaw, M.; Garlan, D., *"Software Architecture: Perspectives on an Emerging Discipline"*. New Jersey, 1996.

[SI01] Sommerville, I., *"Software Engineering"*. 6th ed., Addison-Wesley, 2001.

[ST97] Stroustrup, B., *"The C++ Programming Language"*. 3rd edition, Addison-Wesley, 1997.

[TAS94] Tanir, O.; Servic, S., *"A Standard Simulation Environment: A Review of Preliminary Requirements"*. Proceedings Winter Simulation Conf. Edited by. Swain, G. et al., 1994.

[TO95] Tworzydlo; Oden., *"Knowledge - Based Methods and Smart Algorithms in Computational Mechanics"*. Engineering Facture Mechanics, 60(5/6), 1995.

[VA02] Vieira, M.; Freyry, A.; Santos, F., *"Ferramenta de Visualização de Simulações baseadas no Método do Elemento Finito.* Graduation Work, Center of Informatics, Federal University of Pernambuco, Brazil, 2002.

[VM02] Viceconti, M., *"Replication of Numerical Studies"*. Personal Communication, posted at biomch-l@nic.surfnet.nl. Sent on: March 27, 2002 3:03 pm Subject: BIOMCH-L: BioNet controversial topic #5.

[VM95] Versteeg H.; Malalasekera, W., " An Introduction to Computational fluid Dynamics – The Finite Volume Method", Prentice Hall, England,1995.

[WK99] Warmer, J.; Kleppe, A., *"The Object Constraint Language, Precise Modelling with UML"*. Addison Wesley, 1999.

[WMC95] Workflow Management Coalition, *"The Workflow Reference Model, Workflow Management Coalition Specification"*. Winchester, Hampshire - UK, 1995.

[YB96] Young, K.; Bang, H., *"The Finite Element Method (using Matlab)"*. The Mechanical Engineering Series, 1996.

[YH00] K., Young; B. Hyochoong, *"The Finite Element Method Using Matlab"*. 2th ed., CRC Mechanical Engineering Series, United States, 2000.

[YJ02] Yoder, J.; Johnson, R., *"The Adaptive Object Model Architectural Style"*. Proceeding of The Working IEEE/IFIP Conference on Software Architecture (WICSA3'02), World Computer Congress in Montreal, 2002.

[ZB03] Zeigler, B., "DEVS Today: Recent Advances in Discrete Event-Based Information Technology". 11th ACM / IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, Orlando., 2003.

[ZEI00] Zeigler, B.; Praehofer, H.; Kim, T., *"Theory of Modelling and Simulation – Integrating Discrete Event and Continuous Complex Dynamic Systems"*. Academics Press, San Diego, California, 2000.

[ZL97] Zobrist, G.; Leonard, J., *"Object-oriented Simulation – Reusability, Adaptability and Maintainability"*. IEE Press, 1997.

[ZPK00] Zeigler, B.; Praehofer, H.; Kim, T., *"Theory of Modelling and Simulation"*, 2nd ed., Academic Press, USA, 2000.

[ZP00] Zienkiewicz, O.C.; Taylor R.L., "The Finite Element Method- Solid Mechanics", 5th Editions, Butterworth-Heinemann, Oxford, 2000.

Appendix

A

Examples of Coupled Multi-physic Phenomena Problems

This appendix presents three different examples of problems involving coupled Multi-physics phenomena. First we give a description of a simulator that can be used to solve two of the given problems (example 1 and 2). In addition, in the first example (number 1) we present extra details about exact mathematical models, the differential-algebraic system of equations, and the global algorithm for the problem. These details can help on a more complete understanding of FEM patterns detailed in chapter 5.

Appendix A

A.1 A Simulator Description

An example for a **FEM simulator specification**, which can be used to solve a class of problems exemplified by examples 1 and 2, can be described considering the following global scenario [LS03]: a system capable of solving problems involving transient phenomena; where the phenomena context includes linear temperature-dependent elasticity, rigid body motion and linear heat transfer; Dirichlet restrictions are considered through Lagrange multipliers; for simplification reasons, the simulator process does not include estimation error and adaptation processes (see Figure A-1).

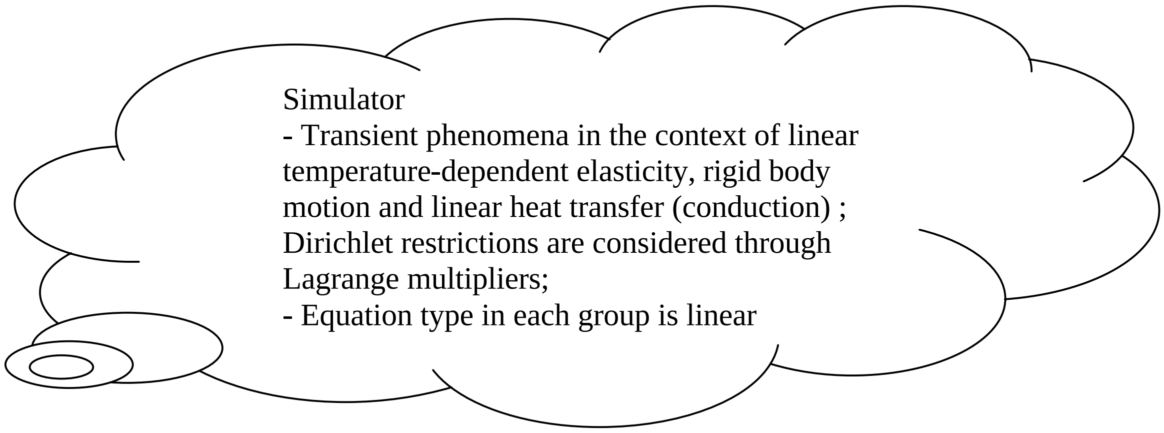

Simulator
- Transient phenomena in the context of linear temperature-dependent elasticity, rigid body motion and linear heat transfer (conduction) ; Dirichlet restrictions are considered through Lagrange multipliers;
- Equation type in each group is linear

Figure A-1 Simulator specification (global scenario)

Example of Blocks and Groups articulation

In any solution algorithm the phenomena are organized in subsets (Groups), each of which are solved independently of the others. This allows for different solution schemes for each Group. The articulation of the solution of Group problems is made by Blocks of Groups. The justification for the Blocks is due to the fact that some Groups of phenomena should be solved before other Groups and the articulation involving some Groups are different from the articulation involving the others.

The specification of the methods used to provide the services offered by each Block depends on its scenario. In the present simulator, the scenario for the solution of each Block considers an iteration between the solution for the Lagrange multipliers (requires stabilization) and the solution for the phenomena themselves. In addition, it assumes that there are two blocks for:

Block 1:

- Group 1: Heat transfer and Group 2: its Lagrange multipliers. The number of heat transfer phenomena depends on the number of simulation regions with this type of phenomenon. The number of Lagrange multiplier phenomena depends on the number of restrictions (Dirichlet boundary conditions in this case) defined for all phenomena of the type heat transfer and

Block 2:

- Group 3: Elasticity, rigid body motion and Group 4: their Lagrange multipliers. The number of elasticity and rigid body phenomena depends on the number of simulation regions with these types of phenomenon. The number of Lagrange multiplier phenomena depends on the number of restrictions (Dirichlet boundary conditions in this case) defined for all phenomena of the types elasticity and rigid body motion.

This type of choice for the number of blocks is due to the fact that the present model of heat transfer does not depend on the result of the elasticity problem. Thus, the heat transfer problem (and respective Lagrange multipliers) can be solved before solving the elasticity/rigid body motion problem (and respective Lagrange multipliers), which in turn depends on the temperature. Moreover, the solution of both blocks are different, because both the resulting semi-discrete equations are different from each other.

It is not mandatory for a problem within the above context to have all types of phenomenon. For instance, it may not have heat transfer phenomena and/or elasticity and/or rigid body motion. However, when the elasticity phenomenon is included, there should be a heat transfer phenomenon defined on the same simulation region, because the elastic material properties depend on the temperature. The heat transfer phenomenon may be as simple as a tool that provides a known temperature distribution but it should be present in order to be coupled to the temperature-dependent elasticity phenomenon. The simulator should be able to take care of the modifications due to the lack of a phenomenon type.

A.1.1 Example 1

The first example is presented in Figure A-2. This example is used to exemplify several parts of this thesis. The problem describes the dynamics of a rigid body attached to an elastic beam, with a temperature dependent constitutive relation, where both are also submitted to thermal loads. Consider the defined geometry consisting of two plane sub-domains Ω_1 and Ω_2. The physical phenomena defined therein are transient state, and include: linear elasticity with temperature-dependent constitutive equations in Ω_1; rigid body motion of Ω_2; linear heat transfer in Ω_1 and Ω_2. The geometry components are: 6 points, 7 curves and 2 plane regions.

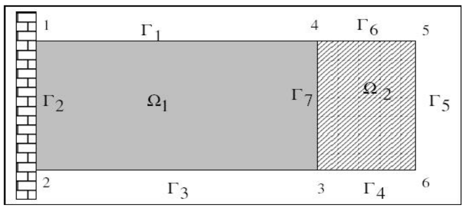


Figure A-2 Coupled Multi-physic - Example 1

The results an engineer may want to obtain and visualize in a case like this may be for example the distribution of the stresses in Ω_1 and temperature distribution in the whole beam structure.

Each one of the described phenomena has its own discrete vector field, geometric domain, couplings with other phenomena, and other relevant data. Further complexities come from the fact that each phenomenon may require its own geometric mesh in spite of the fact that there might be other phenomena defined in the same geometric domain.

Usually a phenomenon has an exact mathematical formulation (behavior laws), comprised of a system of algebraic-differential equations, which govern the behavior of the phenomenon vector field. Restrictions may be applied to the vector field such as boundary conditions, and constitutive restrictions, etc.

A.1.1.1 Exact Mathematical Models

The exact mathematical models are as follows:

i) **Phenomena in** Ω_1: $\mathrm{w}_1 : \Omega_1 \times \Re^+ \to \Re^2$ is the displacement of the points in Ω_1 and T1 $\mathrm{T}_1 : \Omega_1 \times \Re^+ \to \Re$ is the temperature in Ω_1.

i.1) Elasticity (equation of conservation of linear momentum)

$$r_1 \frac{\partial^2 \mathbf{w}_1}{\partial t^2} - \nabla.\mathbf{s}_1(\mathbf{w}_1.\mathrm{T}_1) = 0$$

$$\mathbf{w}_1 = 0 \text{ on } \Gamma_2$$

$$\mathbf{w}_1 = \text{ß}.\mathbf{w}_2 \text{ on } \Gamma_7$$

where $\text{ß} = \begin{bmatrix} 1 & 0 & -d_2 \\ 0 & 1 & d_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \mathbf{e}_1.(\mathbf{k}\times\mathbf{d}) \\ 0 & 1 & \mathbf{e}_2.(\mathbf{k}\times\mathbf{d}) \end{bmatrix}$

with $\mathbf{w}_2 = \begin{bmatrix} u_{M_1} \\ u_{M2} \\ f_M \end{bmatrix}$

with $\mathbf{d} = \begin{bmatrix} d_x \\ d_y \end{bmatrix} = \mathbf{x} - \mathbf{p}_M \quad for \quad \mathbf{x} \in \Gamma_7$

Here s is the stress tensor, r_n is the position of a reference point for Ω_2, and $w_2 : \Omega_2 \times \Re^+ \to \Re^2$ is the vector with the displacement and rotation of the rigid body with respect to the reference point *pM.* The stress tensor depends on w_1 and T_1. $r_1 : \Omega_1 \to \Re$ is the mass density of the material in Ω_1; $n_j : \Gamma_1 \to \Re^2$ is the outward normal to the surface Γ_j, $j = 1,3$.

$$\mathbf{s}_1(\mathbf{w}_1, ?_1).\mathbf{n}_j = 0 \text{ on } \Gamma_j, j = 1,3$$

i.2) Heat transfer (equation of conservation of energy)

$$r_1 c_1 \frac{\partial T_1}{\partial t} - \nabla.(\mathbf{K}_1.\nabla T_1) = 0$$

$$\mathbf{n}_1^T.(\mathbf{k}_1.\nabla T_1) = 0 \quad on \quad \Gamma_2$$
$$\mathbf{n}_j^T.(\mathbf{k}_1.\nabla T_1) = h_j(T_j^\infty - T_1) \text{ on } \Gamma_j, j = 1,3$$

where $k_1 : \Omega_1 \to \Re^2 \times \Re^2$ is the conductivity matrix, $r_1 : \Omega_1 \to \Re$ is the mass density matrix, $c_1 : \Omega_1 \to \Re$ is the specific heat, $n_j : \Gamma_j \to \Re^2$ is the outward normal to Γ_j, $j = 1,2,3$, $h_j : \Gamma_j \to \Re$ and $T_j^\infty : \Gamma_j \to \Re$, $j = 1,3$ are the coefficient of heat transfer by convection and the environment reference temperature for the heat transfer by convection on the boundary Γ_j, $j = 1,3$

$$\mathbf{n}_7^T.(\mathbf{k}_1.\nabla T_1) = q = \mathbf{n}_7^T.(\mathbf{k}_2.\nabla T_2) \quad on \quad \Gamma_7$$

where $\mathbf{n}_7 : \Gamma_7 \to \Re^2$ is the normal to Γ_7 pointing outwards Ω_1, and $\mathbf{g}_7 : \Gamma_7 \to \Re$ is the known heat flux on Γ_7.
Also,

$$T_1 = T_2 \text{ on } \Gamma_7$$

ii) **Phenomena in** W2: $\mathbf{w}_2 : \Omega_2 \times \Re^+ \to \Re^2$ is the displacement of the reference point *pM* and $T_2 : \Omega_2 \times \Re^+ \to \Re$ is the temperature on Ω_2. $\mathbf{w}_2$ does not depends on **x**

ii.1) Rigid body motion

$$?_2.\frac{\partial^2 \mathbf{w}_2}{\partial t^2} = \mathbf{F}_2 + \int_{\Omega_2} \left[\mathbf{k}.\left(\mathbf{d}^{g_2} \times \mathbf{g}_2\right)\right] d\Omega - \mathbf{F}_2^{bd}$$

or

$$?_2.\frac{\partial^2 \mathbf{w}_2}{\partial t^2} = \mathbf{F}_2 + \int_{\Omega_2} \left[\mathbf{k} \times \mathbf{d}^{g_2}\right].\mathbf{g}_2 d\Omega - \mathbf{F}_2^{bd}$$

where $?_2$ is 2×2 matrix of moments of inertia

$$?_2 = \int_{\Omega_2} ?_M \mathbf{ß}^T.\mathbf{ß}\, d\,\Omega,$$

with $\mathbf{g}_2 : \Omega_2 \to \Re^2$ being the mass density defined on Ω_2,

$$\mathbf{F}_2 = \begin{bmatrix} F_{M_1} \\ F_{M_2} \\ M_M \end{bmatrix}$$

is the force vector acting directly on the reference point p_{M}. And, finally,

$$\mathbf{F}_2^{bd} = \int_{\Gamma_7} \mathbf{ß}^{\mathrm{T}}.\mathbf{f}\, d\,\Gamma$$

is the force that domain Ω_1 exerts over Ω_2 through Γ_7. That is,

$$\mathbf{f} = \mathbf{s}_1.\mathbf{n}_7$$

ii.2) Heat transfer

$$r_2 c_2 \frac{\partial \mathrm{T}_2}{\partial t} - \nabla.(\mathbf{K}_2.\nabla \mathrm{T}_2) = \mathrm{R}_2$$

$$\mathbf{n}_j^{\mathrm{T}}.(\mathbf{k}_2.\nabla \mathrm{T}_2) = h_j (T_j^{\infty} - \mathrm{T}_2) \text{ on } \Gamma_j,\, j = 4,5,6$$

$$\mathrm{T}_2 = \mathrm{T}_1 \quad on \quad \Gamma_7 .$$

A.1.1.2 Differential-algebraic system of equations

The differential-algebraic system of equations can be presented in a compact form:

$$\mathbf{M}_e \ddot{U}_e + K_e(T_t) + U_e = f_e \text{ and } \mathbf{M}_t \dot{U}_t + K_t(T_t) = q_t$$

Another more compatible with the solution schemes we are going to use, is the representation below:

$$\begin{bmatrix} M_{ww} & 0 \\ 0 & 0 \end{bmatrix} . \begin{bmatrix} \ddot{\mathbf{W}} \\ \ddot{\boldsymbol{\mu}}_w \end{bmatrix} + \begin{bmatrix} K_{ww} & K_{wm_w} \\ K_{m_w w} & K_{m_w m_w} \end{bmatrix} . \begin{bmatrix} \mathbf{W} \\ \boldsymbol{\mu}_w \end{bmatrix} = \begin{bmatrix} \mathbf{f}_w \\ \mathbf{f}_{m_w} \end{bmatrix}$$

$$\begin{bmatrix} M_{TT} & 0 \\ 0 & 0 \end{bmatrix} . \begin{bmatrix} \dot{\mathbf{T}} \\ \dot{m}_T \end{bmatrix} + \begin{bmatrix} K_{TT} & K_{Tm_T} \\ K_{m_T T} & K_{m_T m_T} \end{bmatrix} . \begin{bmatrix} \mathbf{T} \\ \boldsymbol{\mu}_T \end{bmatrix} = \begin{bmatrix} \mathbf{f}_T \\ \mathbf{f}_{m_T} \end{bmatrix}$$

where

$$\mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix},\ \boldsymbol{\mu}_w = \begin{bmatrix} m_f \\ m \end{bmatrix},\ \mathbf{T} = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix},\ m_T = \lfloor m_q \rfloor$$

A.1.1.3 Global Algorithm for the Problem

For elasticity problem we will adopt the Newmark´s method, which is described as follows:

Algorithm for the global problem:

1) Compute initial states for the heat transfer problem given the initial data T(0)

1.1) Compute Lagrange Multipliers

$$\mathbf{K}_{m_T m_T}\boldsymbol{\mu}_T(0) = \mathbf{f}_{vT}(0) - \mathbf{K}_{mTT}.\mathbf{T}(0)$$

1.2) Compute thermal speed $\dot{\mathbf{T}}(0)$ from:

$$\mathbf{M}_{TT}.\dot{\mathbf{T}}(0) = \mathbf{f}_T - \mathbf{K}_{TT}.\mathbf{T}(0) - \mathbf{K}_{Tm_T}.\boldsymbol{\mu}_T(0)$$

1.3) Compute initial Δt^T and Δt^{m_T}

2) Compute initial states for the elasticity problem given the initial data $\mathbf{W}(0)$ and $\dot{\mathbf{W}}(0)$:

2.1) Compute Lagrange Multipliers $\boldsymbol{\mu}_w(0)$ from:

$$\mathbf{K}_{m_w m_w}(\mathbf{T}(0)).\boldsymbol{\mu}_w(0) = \mathbf{f}_{m_w w}(0) - \mathbf{K}_{m_w w}.\mathbf{W}(0)$$

2.2) Compute acceleration $\ddot{\mathbf{W}}(0)$ from:

$$\mathbf{M}_{ww}.\ddot{\mathbf{W}}(0) = \mathbf{f}_w(0) - \mathbf{K}_{ww}.(\mathbf{T}(0)).\mathbf{W}(0) - \mathbf{K}_{wm_w}.(\mathbf{T}(0)).\boldsymbol{\mu}_w(0)$$

2.3) Compute initial Δt^w and Δt^{m_w}

3) Compute initial $\Delta t = \min\{\Delta t^T, \Delta t^{m_T}, \Delta t^w, \Delta t^{m_w}\}$ and set $t_1 = 0$

4) While $t_1 \leq T_{\max}$ do:

4.0) Set $t_0 = t_1$ and $t_1 = t_0 + \Delta t$

4.1) Solve Heat Transfer problem

4.1.1) Compute

$$\mathbf{f}_T(t_1) = \mathbf{M}_{TT}\left(\frac{2}{\Delta t}\mathbf{T}(t_0) + \dot{\mathbf{T}}(t_0)\right) + \mathbf{f}_T(t_1)$$

4.1.2) Set $\mathbf{T}^0(t_1) = \mathrm{T}(t_0)$ and $m_T^0(t_1) = m_T(t_0)$

4.1.3) Set k=0. While convergence is not achieved do:

4.1.3.0) Compute temperature $\mathrm{T}^{k+1}(t_1)$ from:

$$\left(\frac{2}{\Delta t}\mathbf{M}_{TT} + \mathbf{K}_{TT})\right)\mathbf{T}^{k+1}(t_1) = \mathbf{f}_T(t_1) - \mathbf{K}_{T_{mT}} .\boldsymbol{\mu}_T^k(t_1)$$

4.1.3.1) Compute Lagrange multiplier $\boldsymbol{\mu}_T^{k+1}(t_1)$ from:

$$\mathbf{K}_{m_T m_T} .\boldsymbol{\mu}_T^{K+1}(t_1) = \mathbf{f}_{m_T}(t_1) - \mathbf{K}_{m_T T} .\mathbf{T}^{K+1}(t_1)$$

4.1.3.2) Compute error for convergence test based on

$$\left\|\mathbf{T}^{K+1}(t_1) - \mathbf{T}^K(t_1)\right\|_w \text{and} \left\|\boldsymbol{\mu}_T^{k+1}(t_1) - \boldsymbol{\mu}_T^k(t_1)\right\|_{L_{gm}}$$

4.1.3.3) Set k=k+1

4.1.4) Accept solution as $\mathbf{T}_1^{K+1}(t_1)$ and $\boldsymbol{\mu}_T^{k+1}(t_1)$ as the solution instant t_1 for the heat transfer and Lagrange multiplier problems. Store them.

4.1.5) Compute $\dot{\mathbf{T}}(t_1)$

$$\dot{\mathbf{T}}(t_1)\left(\frac{2}{\Delta t}(\mathbf{T}(t_1) - \mathbf{T}(t_0))\right) - \dot{\mathbf{T}}(t_0)$$

4.1.6) Compute next Δt^T and Δt^{m_T}

4.2) Solve Elasticity Problem

4.2.1) Compute

$$\tilde{\mathbf{W}}(t_1) = \mathbf{W}(t_0) + \Delta t\,\dot{\mathbf{W}}(t_0) + \frac{\Delta t}{2}(1 - 2b\ )\ddot{\mathbf{W}}(t_0)$$

$$\dot{\tilde{\mathbf{W}}}(t_1) = \dot{\mathbf{W}}(t_0) + \Delta t(1 - g\ )\ddot{\mathbf{W}}(t_0)$$

and

$$\tilde{m}_w^0(t_1) = \boldsymbol{\mu}_w(t_0)$$

4.2.2) Set $\mathbf{W}^0(t_1) = \mathbf{W}^0(t_0)$ and $\boldsymbol{\mu}_w^0(t_1) = \boldsymbol{\mu}_w(t_0)$

4.2.3) Set k=0. While convergence is not achieved, do:

4.2.3.0) Compute $\ddot{\mathbf{W}}^k(t_1)$ from

$$\left(\mathbf{M}_{ww} + b\ \Delta t^2 \mathbf{K}_{ww}(\mathbf{T}(t_1))\right)\ddot{\mathbf{W}}^{k}(t_1) = \mathbf{f}_w(t_1) -$$

$$-\mathbf{K}_{ww}(\mathbf{T}(t_1)).\tilde{\mathbf{W}}(t_1) - \mathbf{K}_{wm_w}(\mathbf{T}(t_1)).\tilde{m}_w^{k}(t_1)$$

4.2.3.1) Compute

$$\mathbf{W}^{k+1}(t_1) = \tilde{\mathbf{W}}(t_1) + b\ \Delta t^2\, \ddot{\mathbf{W}}^k(t_1)$$

and

$$\dot{\mathbf{W}}^{k+1}(t_1) = \dot{\tilde{\mathbf{W}}}(t_1) + g\ \Delta t\, \ddot{\mathbf{W}}^k(t_1)$$

4.2.3.2) Compute $\boldsymbol{\mu}_w^{k+1}(t_1)$ from:

$$\mathbf{K}_{m_w m_w}.\boldsymbol{\mu}_w^{k+1}(t_1) = \mathbf{f}_{m_w}(t_1) - \mathbf{K}_{m_w w}(\mathbf{T}(t_1)).\mathbf{W}_k + 1(t_1)$$

4.2.3.3) Set $\tilde{\boldsymbol{\mu}}_w^{k+1}(t_1) = \boldsymbol{\mu}_w^{k+1}(t_1)$

4.2.4) Compute error for convergence test based on

$$\left\|\mathbf{W}^{K+1}(t_1) - \mathbf{W}^{K}(t_1)\right\|_w \text{and} \left\|\boldsymbol{\mu}_w^{k+1}(t_1) - \boldsymbol{\mu}_w^{k}(t_1)\right\|_{L_{gm}}$$

4.2.3.5) Set k=k+1

4.2.4) Accept solution $\mathbf{W}_1^{K+1}(t_1)\,\text{and}\,\boldsymbol{\mu}_w^{k+1}(t_1)$ as the solution at instant t_1 for the elasticity and Lagrange multiplier problems. Store them.

4.2.5) Compute next Δt^w and Δt^{m_w}

4.3) Compute $\Delta t = \min\{\Delta t^T, \Delta t^{m_T}, \Delta t^w, \Delta t^{m_w}\}$

A.1.2 Example 2

The second example of a simulation problem is presented in Figure A-3 [LS03]. It is composed of three rigid bodies inserted into an elastic bounded region, with no heat transfer. The geometry of the problem is a square plane elastic region with three rigid bodies inserted in it. Two of the rigid bodies are only partially inserted in the elastic region, while the third one is in its interior. The rigid body Ω_2 will receive a sudden discharge of energy on Γ_5, modelled by an initial nonzero velocity, while all the system has zero initial displacement. The other bodies have zero initial velocity, with the exception of the elastic interface with Ω_2. The free surfaces Γ_j, j = 4,6,7,8, has zero

Neumann conditions. Each rigid body has its own point of reference and other geometric properties.

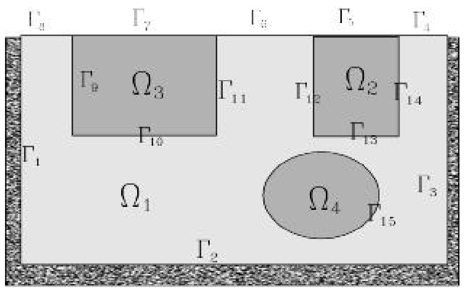


Figure A-3 Coupled Multi-physic Example 2

The linear elasticity phenomena in Ω_1 will be denoted by its vector field w_1, and the rigid body motion of body Ω_j will be denoted by its vector field w_j, j = 2,3,4.

All vector fields will be governed by the equation of motion (elastic and rigid bodies) (with the exception of temperature dependence, for which will be given a reference temperature). There will be 4 phenomena: elasticity and three rigid body phenomena.

A simulator very similar to the previous one can be used in this problem, which is different from the other geometrically, but not in its essence: the phenomena set and solution scheme can remain in the same context.

A.2 Example 3

Another problem is shown in Figure A-4 [LSR02b]. Let Ω_0 be a cavity filled with a Newtonian incompressible fluid, which is driven by a piston represented by the moving region Ω_4. The evolution of Ω_4 is defined by its known movement as a rigid body, and restricted by the rigid walls Γ_8 and Γ_{10}. The boundary lines Γ_{11} and Γ_9 cannot go beyond the lines 1-7 and 13-14, respectively.

The material of regions Ω_1, Ω_3 and Ω_5 are elastic and the others are not. Ω_2 is an insulated rigid region, which has no phenomena defined inside it. Region Ω_5 coincides with the boundary portion Γ_{13} and is a heat generator. Thus, the heat will be conveyed by the fluid and transferred to the outside through Ω_1 and its boundary Ω_3 and through Ω_4 and its boundary Γ_9. All blue boundary lines are rigid and insulated. Red boundary lines are interfaces between the fluids and are either elastic or moving regions and are not insulated. The displacement of Ω_4 will be considered to be small in some sense.

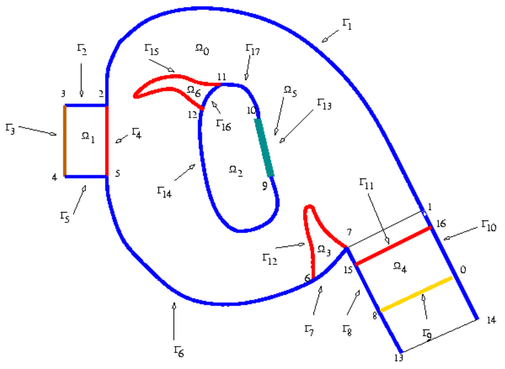


Figure A-4 Coupled Multi-physic Example 3

As can be seen, 16 points, 17 curves and 7 plane regions define the geometry. The phenomena defined for this model problem are the flow of an incompressible Newtonian fluid with heat transfer in $\Omega_{0;}$ elasticity and heat transfer in Ω_i, i = 1, 3, 6; heat transfer and constrained movement of a rigid body in Ω_4; heat generation and transfer in 1-D with lateral heat exchange by convection with the fluid in $\Omega_{6.}$

A simulator can help, for example, in the evaluation of the distribution of the involved phenomena (pressure, fluid velocity, temperature, stress) in the defined geometry

Appendix

B

Approaches for Simulation Software Development

This appendix presents several ways to develop simulation systems, including some examples of general-purpose languages, simulation languages and simulation environments. The appendix includes approaches that are specific and non-specific to FEM simulations.

B.1. Simulation Software Approaches

There are different approaches for the development of simulation systems. In the sequence we present some approaches, which deal with several kinds of simulation, they include:

- **General Purpose Languages** in the original form, where the user has to build the whole program. Examples of these languages are FORTRAN, C, and C++, etc. This approach is also called Do-it-yourself. The main features of this approach are: (a) The existence of the pre-existing resources and programmers, resulting in low costs in the development. (b) The elaboration of simulators being extremely specialized, for which existing packages do not support appropriate facilities and functionalities; (c) The integration with other pre-existing programs. Some disadvantages are the involved time and cost in the development of the program, which can invalidate the economy originally expected; and the need of high ability in the programming expertise, which could be not available or can became unavailable in the future.
- **Pre-built libraries**, where routines are used to join programs. Examples are the GASP, SIMON, DIFFPACK[LAN97]and PZ [DP99, DP04] that is used on finite element method simulations. The pre-built libraries are a natural complement of the previous approach (Do-it-yourself). Other examples are pre-built libraries in FORTRAN, C, and C++, etc. Specially attention must be given to FORTRAN pre-built libraries, which are largely used in numerical methods and are highly consolidated.They allow saving in time guarantying the main advantages of the previous approach. Some disadvantages are relative to the time and cost required for the program development, the need of a specialist, and the difficulty in reusing code or libraries, which require good documentation.
- **Simulation Languages**, where the programming is done in a suitable syntax. Examples are SIMULA [PJ70] (pioneer language), SIMAN [PSS90], SIMSCRIPT [REC93]; and Simple++ [KL97]. Some of the common aspects and supported features are: (a) the control of the main aspects of event agenda maintenance and the sequencing of model operations; (b) an appropriate syntax for systems modelling; (c) facilities for the trace and data acquisition in the simulation execution; (d) support for the experimentation and results analysis (e) Facilities for modelling and reduction of time in the program development. Disadvantages include the need to acquire the appropriate system; requirements of training and the permanent support offered by the systems fabricants; and the question of high-specialized employee for developing the models.
- **Programs generators**, systems based on interactive graphic interfaces, for example, ARENA [FP01] (for SIMAN simulation language), PROMODEL [BBG97] (used on manufactured oriented software). This approach complements the idea implemented by the diagrams, allowing the representative code generation of the model. The source code generated can be changed, compiled or interpreted later. This approach is based on diagrams of activity cycles. It executes consistency tests in the model, and has complementary facilities for tracking simulation

running, allowing data acquisition and results analysis. Disadvantages include: the required ability from the programmer with the generated source code; due to the use of activities cycle diagrams, it creates a stereotyped model of the system, presenting flexibility also restricted to a class of models, in spite of allowing the addition and maintenance.

- **Simulation Environments:** They offer extra facilities for simulation execution based on models animated by interactive models of systems. The graphic use is complementary to the modelling process, the flexibility is maintained through the use of a language with a specific syntax for modules description. They execute consistency tests of the model, and also support "debug" facilities. They facilitate the verification of models and their validation. Experimentations can also be implemented in an interactive way, and their results can be analysed in graphics, helping in the experiments design. Additionally, they provide complementary facilities for the tracking of simulation running, allowing the data acquisition and results analysis. Disadvantages include: the need to understand, the required effort and great ability of the programmer to elaborate the complex models. Some examples of general simulation environments include DEVSIM++, MATLAB [MAT04], SCIRUN [PWC97], Matematica, Mapple and Khoros[KHR04].

In the next section we detail some existing simulation environments.

B.2. Simulation Environments

This section gives a short description of some environments for scientific computation that support simulation and are extensively used. We can classify them as: general purpose programing environment (e.g. MATLAB), domain specific rigid environments (e.g. ANSYS) and domain specific flexible environments (e.g FEMLAB).

Next we will give a brief description of them:

- ANSYS [ANS04], is a high reliabile and sofisticated packages specific for FEM simulations (supporting phenomena coupling) and is used across a broad spectrum of industries. These environments include pre, post and analysis steps of a simulation experiment development. ANSYS provides an engineering simulation software that incorporates design simulation and virtual prototyping into a product development process. With the ANSYS Software Suite, the user can determine real world structural, thermal, electromagnetic and fluid-flow behaviour of 3-D product designs, including the effects of multiple physics when they are coupled together for added accuracy and reliability.
- MATLAB is a technical computing environment [MAT04], which offers a set of integrated products for data analysis, visualization, application development, simulation, design and code generation. It is used extensively for rapid algorithm research development (model-based) and system-level simulations, and is highly applied for FEM simulations. It integrates the SIMULINK, a simulation and prototyping tools for modeling, simulating, and analyzing real world, dynamic systems. Simulink provides a block diagram interface that is built on the core of MATLAB numeric, graphics and programming functionality. MATLAB supports

toollboxes, which are collections of highly-optimized, application specific functions that extends MATLAB and Simulink. Toolboxes suport applications involving signal and image processing, control system design, optimization, finacial engineering, symbolic math and neural works, etc. Tool box functions are built in the MATLAB language and can be easily viewed and modified.

- FEMLAB [FEL04] is an interactive environment for modeling and simulating scientific and engineering problems based on partial differential equations. This environment was detailed before in chapter 2.
- SCIRUN, a scientific programming environment that allows the interactive construction, debugging and steering of large-scale scientific computations [PWC97]. This environment was detailed before in chapter 2.
- DEVSim++ environment allows modelers to develop discrete event simulation models using the hierarchical composition methodology in an object-oriented framework. The environment combines the DEVS (Discrete Event System Specification) formalism with object-oriented paradigm [KA97]. The DEVS is a formalism for discrete event modelling and simulation, which provides a means of specifying a mathematical object called a system. The insight provided by the DEVS formalism is in the simple way that it characterizes how discrete event simulation languages specify discrete event system parameters. Having this abstraction, it is possible to design new simulation environments with sound semantics that have a number of benefits to be described [ZB03]. The formalism specifies direct-events models in a hierarchical, modular form. Within the formalism one must specify the basic models from which the larger ones are built and how these models are connected together in a hierarchical fashion. A basic model, called atomic model, has the dynamics of the model. The reusability of the framework is based on: (a) reusability of functions in the development of models; and (b) reusability of models for the development of new models that are slightly different from the old ones.
- Khoros [KHR04] is an integrated software development environment for information processing and visualization. It is used as both a scientific problem-solving environment and as a software development and integration environment. Khoros provides: (i) Solutions for scientific and engineering problems; (ii) A Visual Programming Environment for Solution Creation and Problem Solving; (iii) Software Development Environment and Tools; (iv) Libraries for Portability, Scalability, Data Access, GUI creation, Data Visualization; (v) Visualization Applications. Khoros is made up of different programs and libraries organized into toolboxes, which contain different capabilities; toolboxes may be mixed and matched according to the user's needs. Khoros supports solutions for common engineering and scientific data analysis and visualization problems. It offers operators facilitate problem solving in a wide variety of application domains used in research, science, government and industry. All information processing and visualization programs in Khoros are available via Cantata, which is a graphically expressed, data flow visual language which provides a visual programming environment within the Khoros system. This visual language provides support for simulation and prototyping system.

Appendix

C

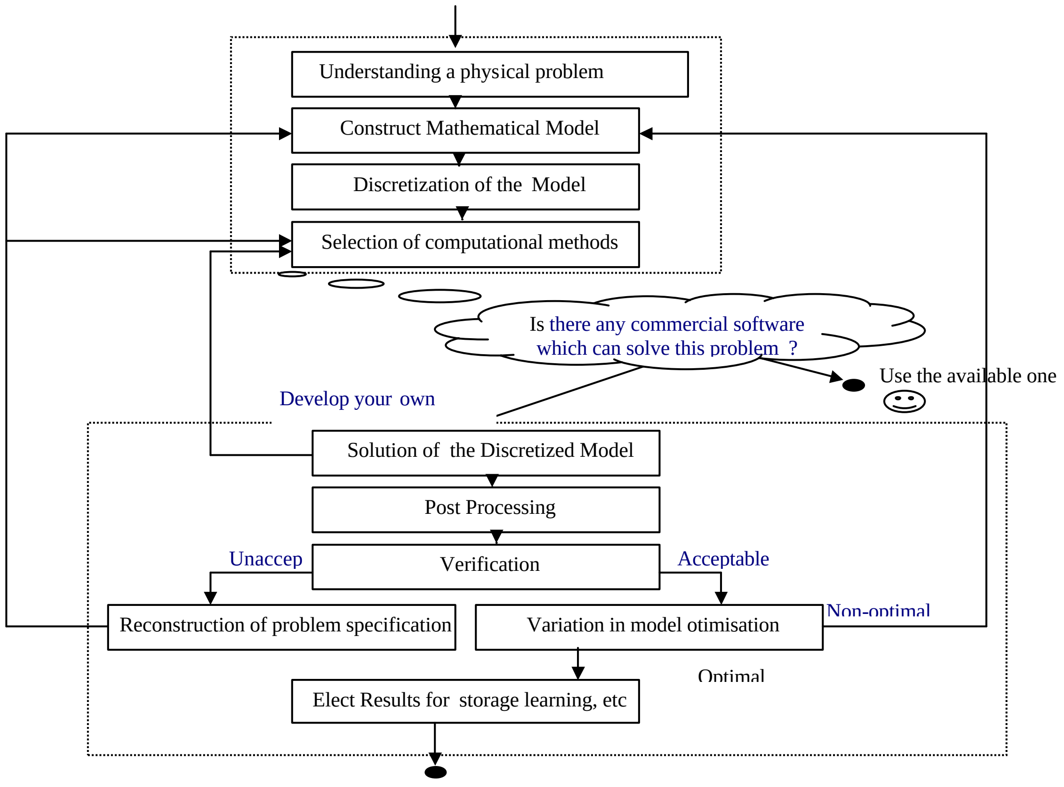
The Plexus Simulation Environment Interface

This appendix gives a brief explanation about the Plexus system and also shows its interface.

C.1. Introduction

When a problem must be solved using FEM simulation, the designer will understand the physical problem, construct the mathematical model, discretize its model and select the computational methods to be applied. At this time, the designer has, in a simplified view, two ways of solving the problem: use existing software, such as the ones described in appendix B or develop one.

There are several approaches to develop a simulation, such as general purpose languages, simulation languages and simulation environments. This work proposed the Plexus environment for the development of simulators specific for the development of simulation of coupled multiphysic phenomena based on the FEM. This environment allows for the solution of simulations that are in interdisciplinary areas of mechanics such as stress and termal analysis.



Understanding a physical problem
Construct Mathematical Model
Discretization of the Model
Selection of computational methods
Is there any commercial software which can solve this problem ?
Use the available one
Develop your own
Solution of the Discretized Model
Post Processing
Unaccep
Verification
Acceptable
Reconstruction of problem specification
Variation in model otimisation
Non-optimal
Optimal
Elect Results for storage learning, etc

Figure C-1 Simulation Process for Problem Solving

C.2 Plexus Environment Interface

Figure C-2 represents on of the main Plexus System processes, the *Administration/ System Loading*, which supports the system management and loading of general system data and basic data type tables, the simulator building/configuration.

Figure C-3, represents the other main processes: *Pre-processing*, where the user inputs the problem data, and where dynamic structures for simulation are built; *Simulation Processing*, where data are processed to obtain the solution and where verification occurs; *Post-processing*, where the solution is processed in order to obtain the quantities of interest for the user and for the required visualization. This component also deals with system validation.

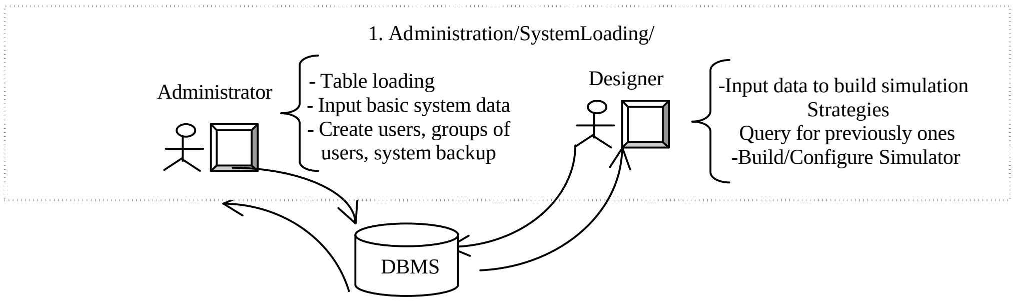

1. Administration/SystemLoading/
Administrator
- Table loading
- Input basic system data
- Create users, groups of users, system backup
Designer
-Input data to build simulation Strategies
Query for previously ones
-Build/Configure Simulator
DBMS

Figure C-2 System Overview (Administration)

In Plexus we can think of a process (method) that helps to generate configurable simulators that represent different types of simulators (e.g.for a type which considers time dependent simulators). Then again, each simulator is capable of running through several kinds of proposed problems.

The proposed interface helps to answer the following questions: What are the specific steps of the proposed method? What kinds of simulators are available? How can we register a new one? What are the steps in a new Simulator creation? How to select a kind of simulator? How to build a specific kind of simulator? How to configure the simulator? What raw data must be supplied for a problem to be solved?

To support the high level of abstraction, flexibility, reusability, and data security, available in the Plexus environment, there is a repository manager, which maintains the general abstract data related to the context, the algorithms that take part in different simulation strategies, the simulation problem's data and also the simulation's intermediary data and results. Bellow we present some of the main window interfaces.

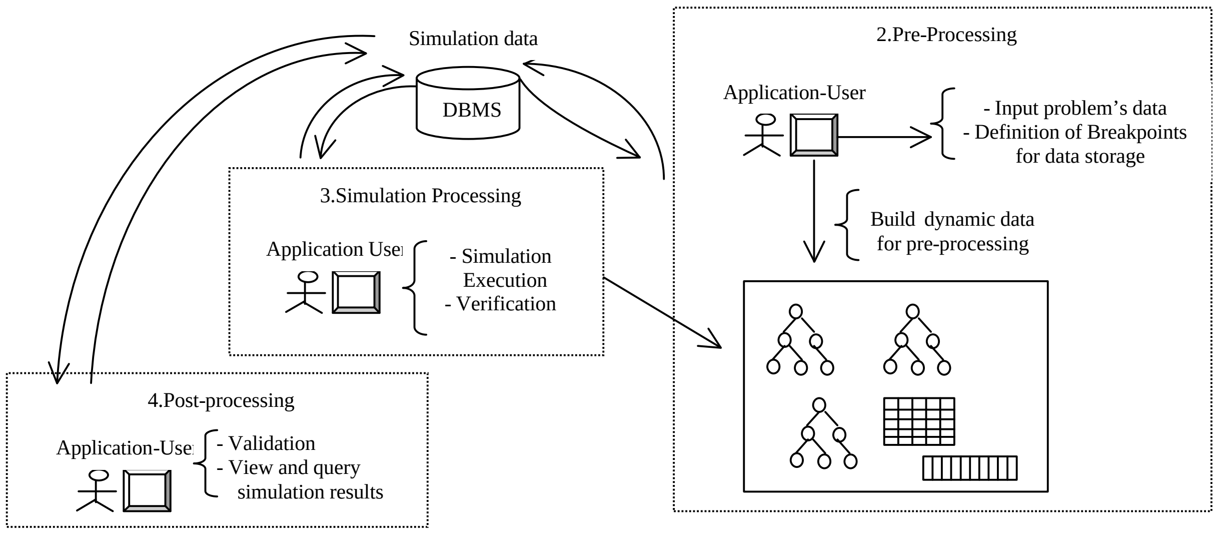


Figure C-3 Plexus System Overview (Simulation Components)

C.3 Plexus Use Cases

This section presents Plexus use cases, see Figures C-4 to C-7. Details about the involved users and package information were described in chapter 3.

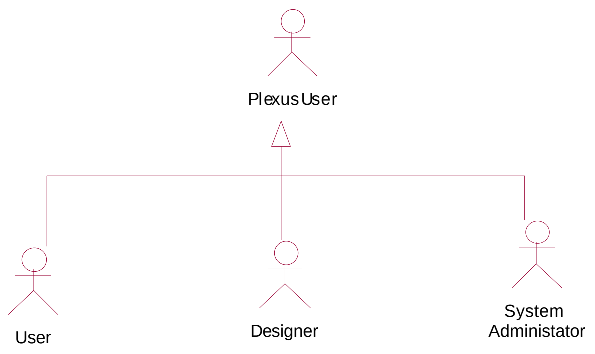


Figure C-4 Existing actors

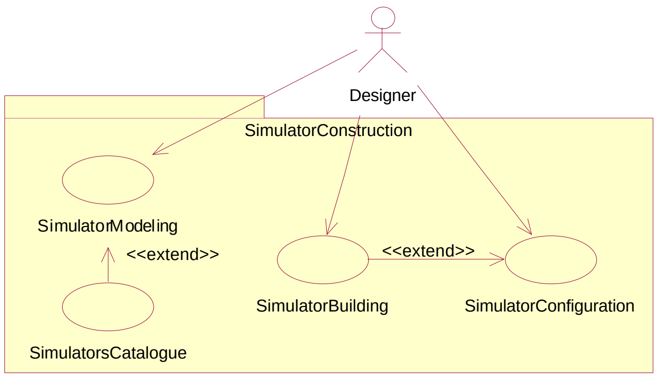


Figure C-5 Simulator Construction Use Case

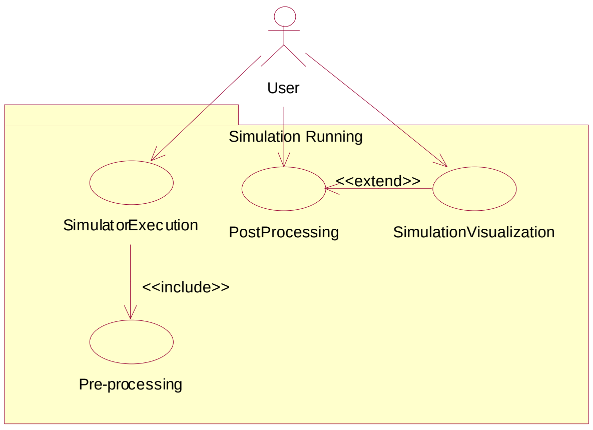


Figure C-6 Simulation Running Use Case

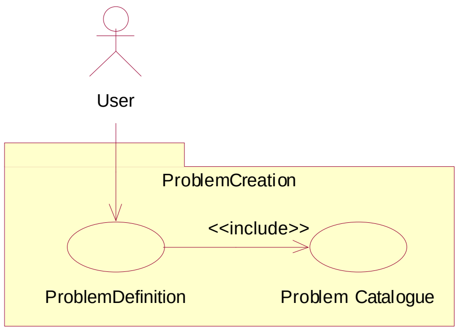


Figure C-7 Problem Creation

C.4 Plexus System Windows

Figure C-8 presents the Plexus main window. The “Register” option will be responsible for registering basic information. This includes data that can be used in simulator definition and also in simulation problems definition. Information such as integration rules and shape functions are stored as code.

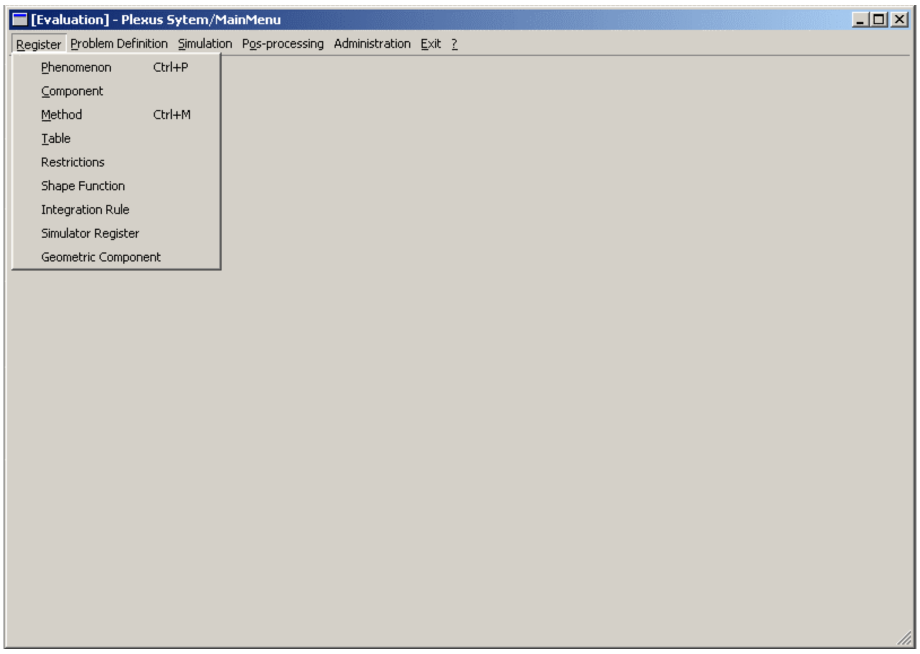
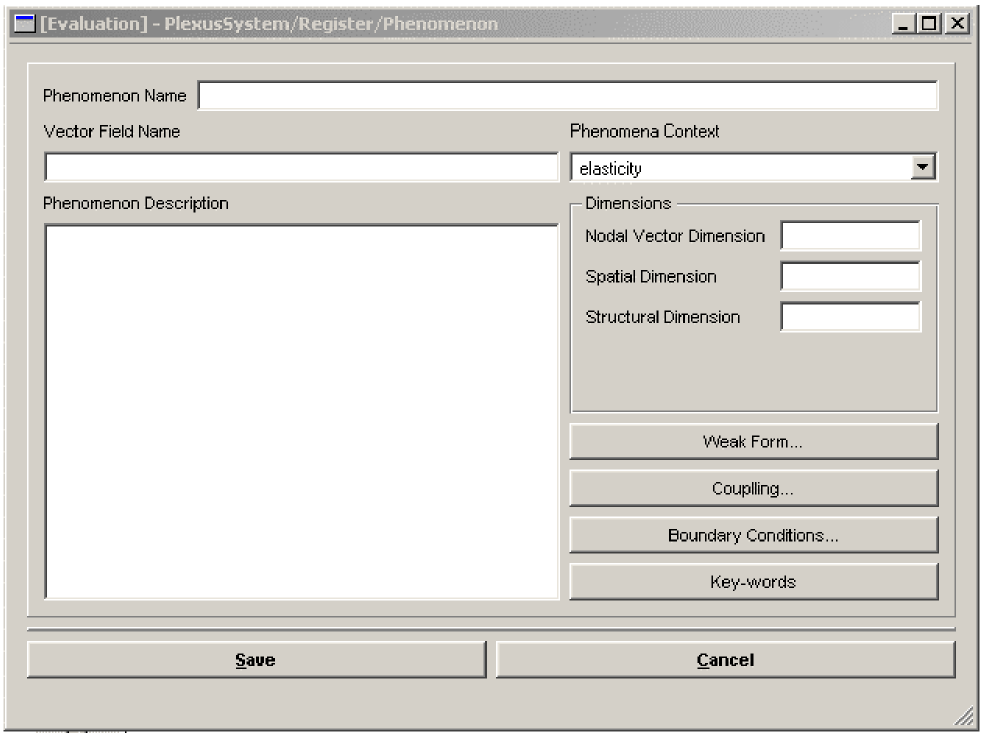


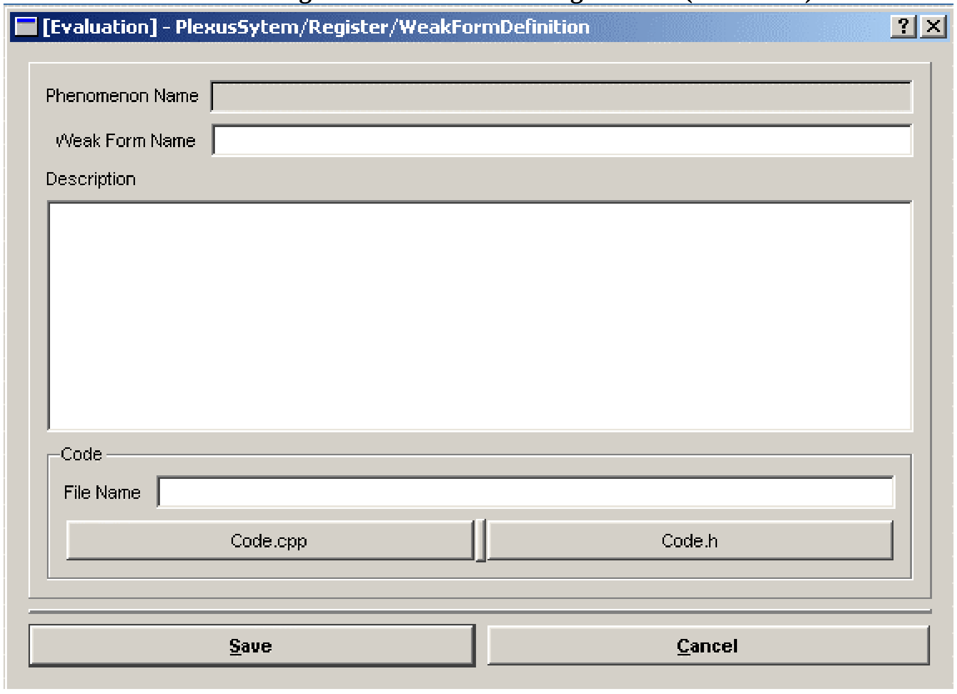
Figure C-8 Plexus Main Window

Figure C-9 presents the generic Phenomenon registering, which can be further reused in the simulation problems definition. There are several pieces of information defined for a phenomena, which include a weakform, see Figure C-10, which is a piece of code associated to the respective phenomena.


[Evaluation] - PlexusSystem/Register/Phenomenon

Phenomenon Name	
Vector Field Name	
Phenomena Context	elasticity
Phenomenon Description	
Nodal Vector Dimension	
Spatial Dimension	
Structural Dimension	

Figure C-9 Phenomena Registration (basic data) Window


[Evaluation] - PlexusSytem/Register/WeakFormDefinition

Phenomenon Name	
Weak Form Name	
Description	
File Name	

Figure C-10 Phenomenon-Weak Form Registration

The simulation model is described during the simulation registering. As described in chapter 3, in the simulation model the user must define the Global Scenario, which is composed of several options (see Figure C-12), the Global Skeleton which is associated with the simulator model. It can be suggested by the interface or defined by the user (Figure C-13) and the Phenomena Context (which must be selected from the one previously defined in a specific table of Phenomena context).

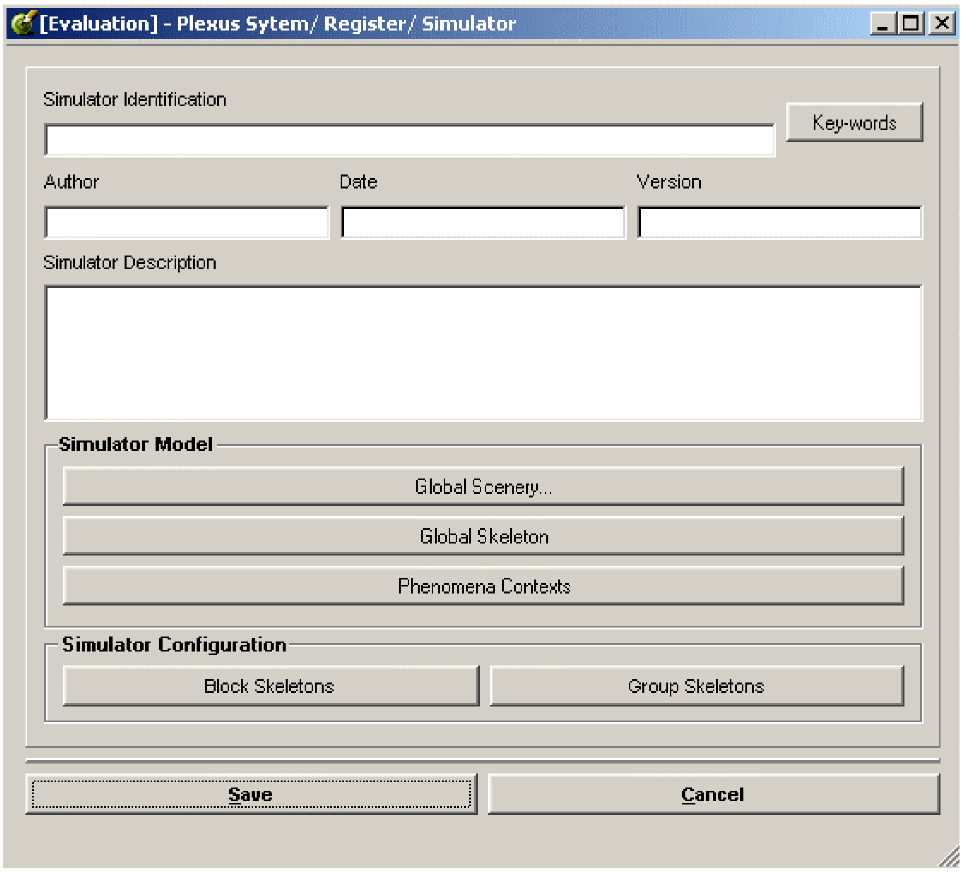


Figure C-11 Simulator Register

During the simulator registration, the simulator configuration is also available. There are default values or they can be defined afterwards.

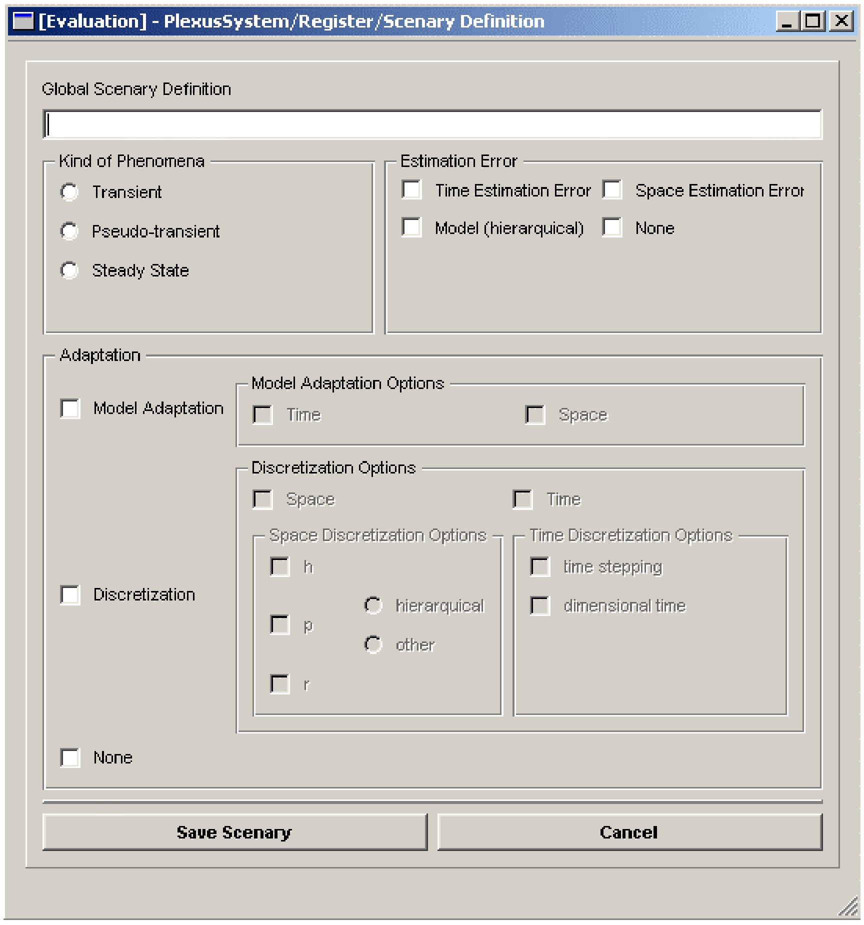
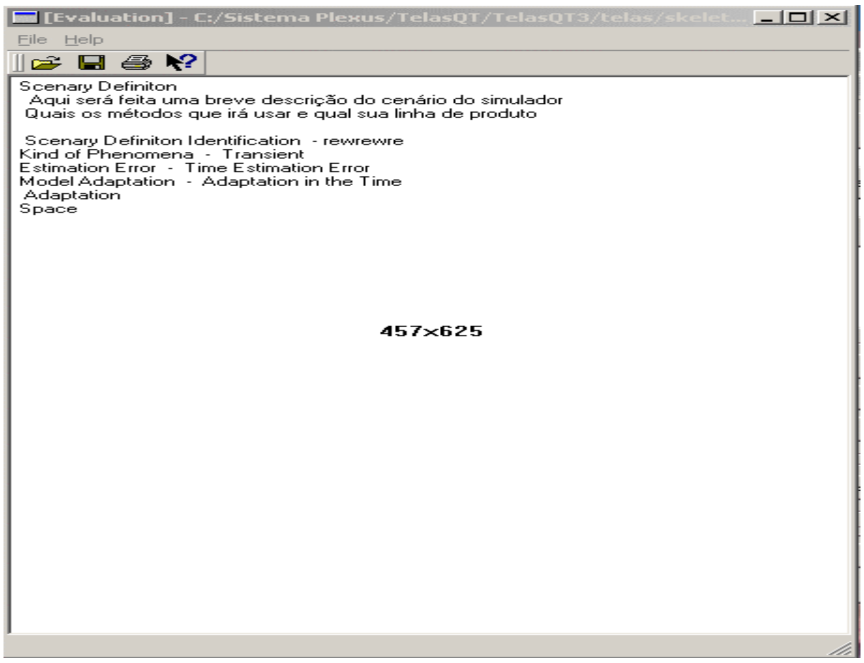


Figure C-12 Global Scenario


[Evaluation] - C:/Sistema Plexus/TelasQT/TelasQT3/telas/skelet...
File
Help
457x625

```
Scenary Definiton
 Aqui será feita uma breve descrição do cenário do simulador
Quais os métodos que irá usar e qual sua linha de produto

 Scenary Definiton Identification  - rewrewre
Kind of Phenomena  -  Transient
Estimation Error  -  Time Estimation Error
Model Adaptation  -  Adaptation in the Time
 Adaptation
Space
```

Figure C-13 Global Skeleton Code

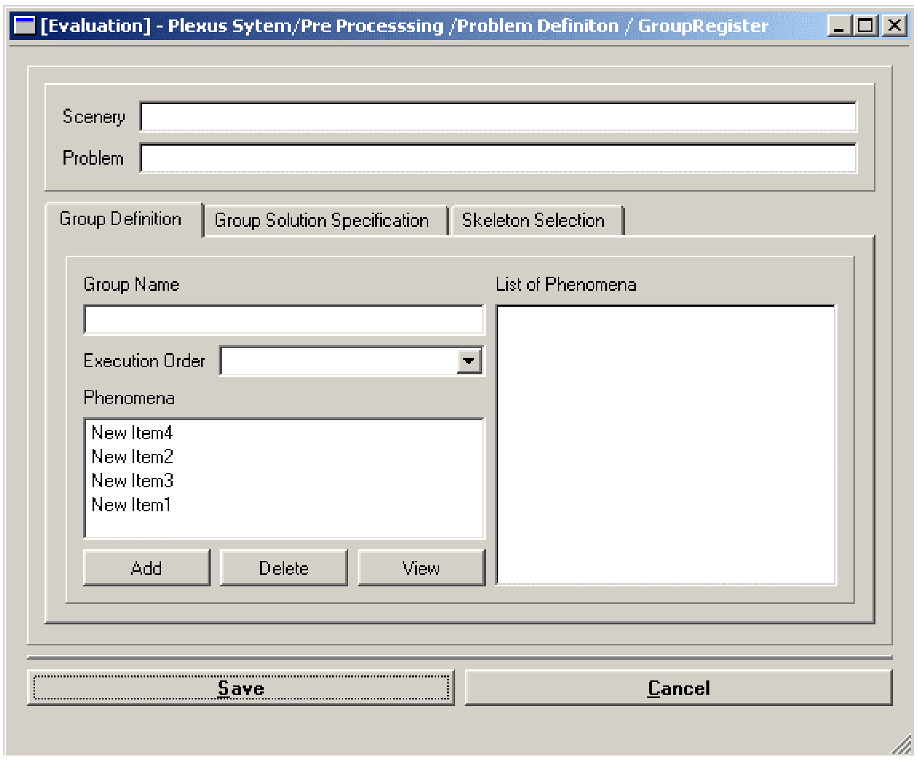
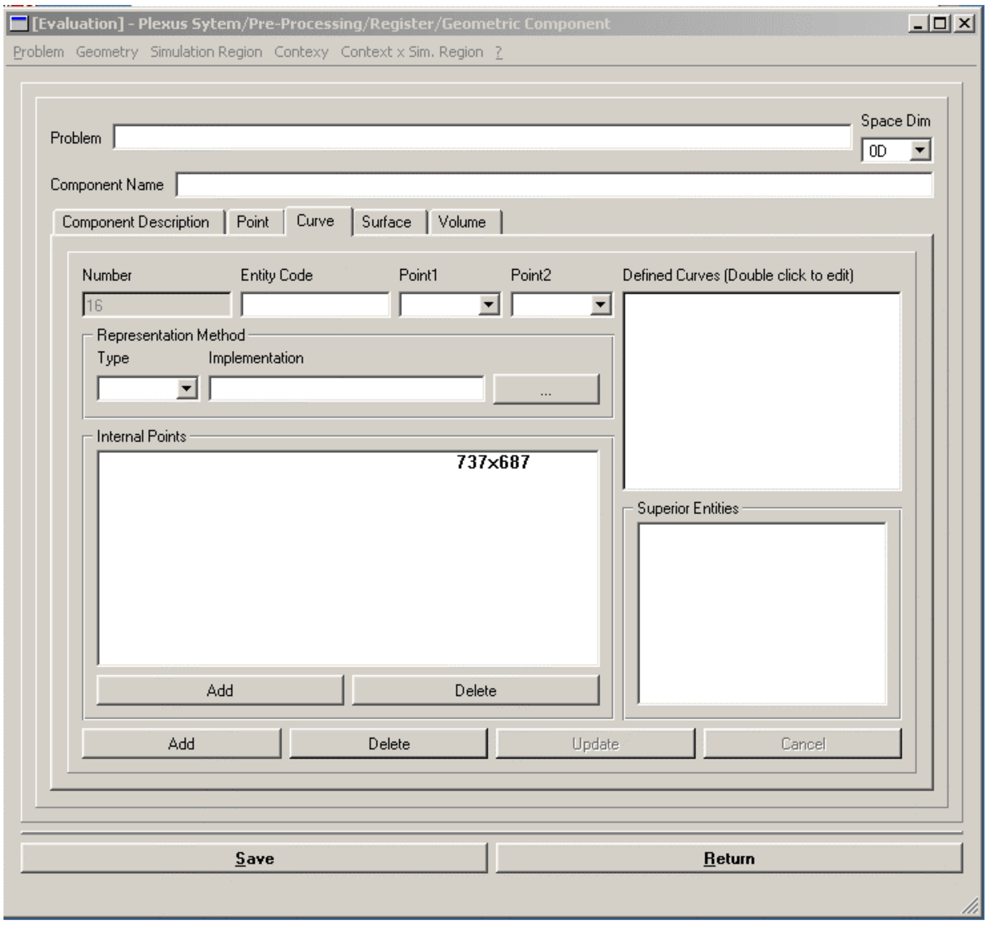


Figure C-14 Group Definition


[Evaluation] - Plexus Sytem/Pre-Processing/Register/Geometric Component
Problem Geometry Simulation Region Contexy Context x Sim. Region ?
Add
Delete
Add
Delete
Update
Cancel
Save
Return
737x687

Problem

Space Dim 0D

Component Name

Component Description | Point | Curve | Surface | Volume

Number 16

Entity Code

Point1

Point2

Defined Curves (Double click to edit)

Representation Method

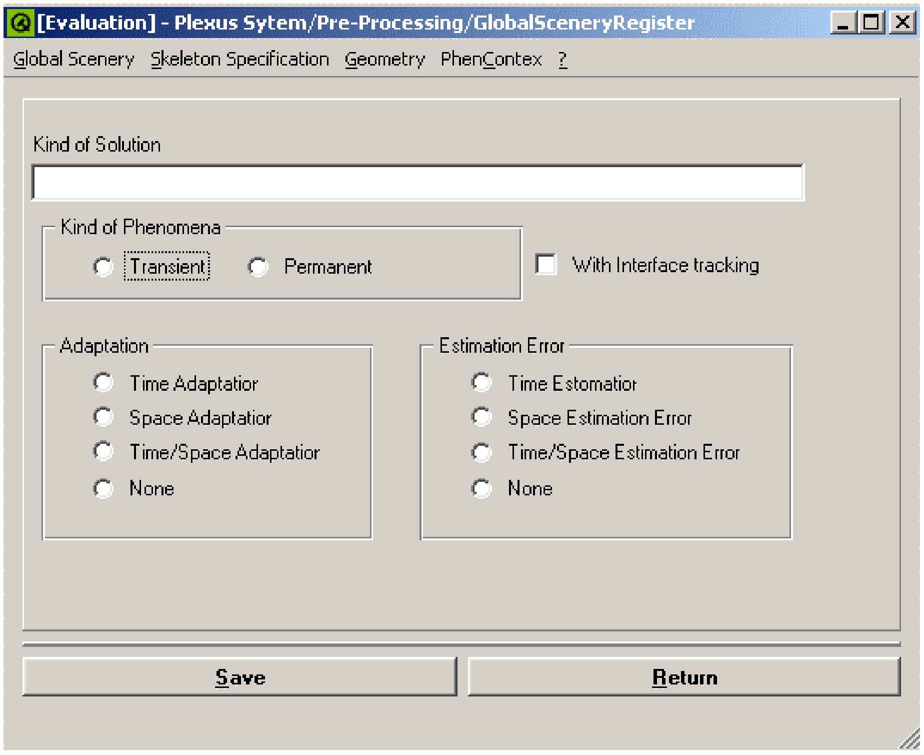
Type

Implementation

Internal Points

Superior Entities

Figure C-15 Geometry Registration Window


[Evaluation] - Plexus Sytem/Pre-Processing/GlobalSceneryRegister
Global Scenery Skeleton Specification Geometry PhenContex ?
Save
Return

Kind of Solution

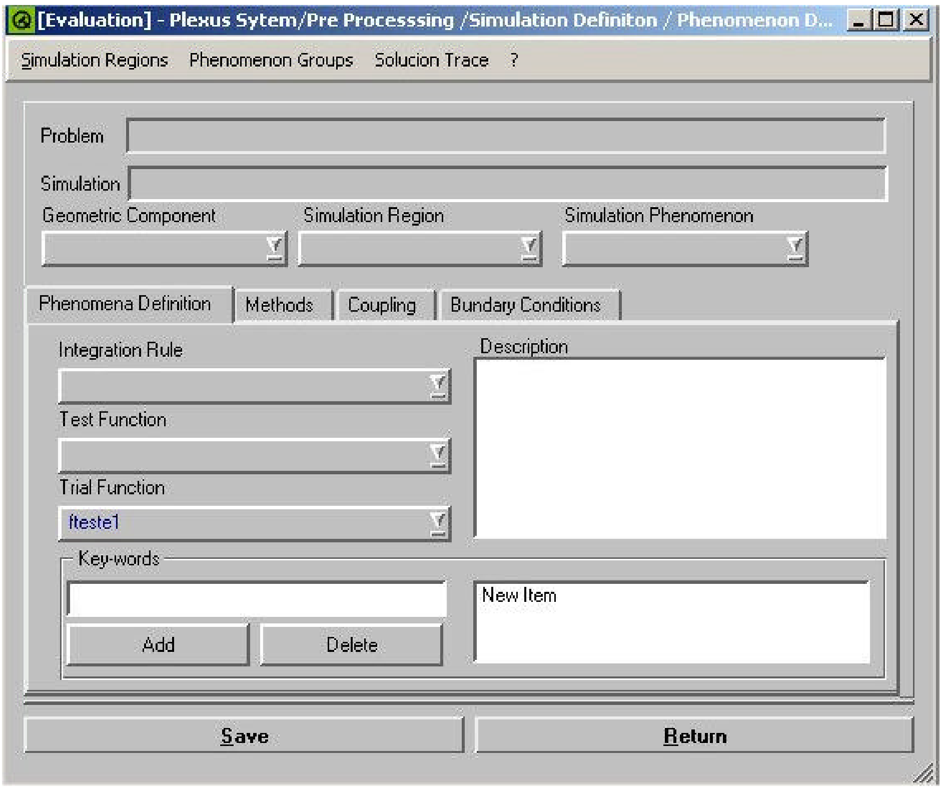
Kind of Phenomena: Transient / Permanent

With Interface tracking

Adaptation: Time Adaptatior / Space Adaptatior / Time/Space Adaptation / None

Estimation Error: Time Estomatior / Space Estimation Error / Time/Space Estimation Error / None

Figure C-16 Simulator Scenario Window


[Evaluation] - Plexus Sytem/Pre Processsing /Simulation Definiton / Phenomenon D...
Simulation Regions Phenomenon Groups Solucion Trace ?
Phenomena Definition
Methods
Coupling
Bundary Conditions
Add
Delete
Save
Return

Problem

Simulation

Geometric Component

Simulation Region

Simulation Phenomenon

Integration Rule

Test Function

Trial Function: fteste1

Description

Key-words

New Item

Figure C-17 Simulation Problem Phenomena Window

Tese de Doutorado apresentada por **Maria Lencastre Pinheiro de Menezes e Cruz Dourado de Azevedo** a Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Conceitualização de um ambiente para o Desenvolvimento de Simuladores Baseado no Método do Elemento Finito"**, orientada pelo **Prof. Jaelson Freire Brelaz de Castro** e aprovada pela Banca Examinadora formada pelos professores:

[illegible]
Prof. Alexandre Marcos Lins de Vasconcelos
Departamento de Sistemas de Computação – CIn / UFPE

[illegible]
Profa. Judith Kelner
Departamento de Informação e Sistemas – CIn / UFPE

[illegible]
Profa. Itana Maria de Souza Gimenes
Departamento de Informática / UEM

[illegible]
Prof. José Maria Andrade Barbosa
Departamento de Engenharia Mecânica / UFPE

[illegible]
Prof. João Baptista da Silva Araújo Junior
Departamento de Informática / Universidade Nova Lisboa

Visto e permitida a impressão
Recife, 27 de fevereiro de 2004.

[illegible]
Prof. JAELSON FREIRE BRELAZ DE CASTRO
Coordenador da Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco.