Chapter 1

Introduction

The automotive, aerospace and building sectors have traditionally used simulation programs to improve their products or services, focusing their computations in a few major physical process: fluid dynamics, structural, metal forming, etc. Nevertheless, the new needs for safer, cheaper and more efficient goods together with the impressive growth of the computing facilities, demand an equivalent solution for multi-disciplinary problems. Moreover, new applications in the food, chemical and electromagnetic industry, among others, are not useful at all without a multi-disciplinary approach.

Some illustrative cases can be found in the design of sails and sailboats where structural and fluid dynamic computations have to be taken in account, sterilization processes (thermal and fluid dynamic analysis), strong magnet design (structural, thermal and magnetic analysis) or photovoltaic cells (thermal and electrical computations), among many others.

The increasing importance of solving multi-disciplinary problems makes developers put more attention to these problems and deal with difficulties involved in developing software in this area.

1.1 Motivation

The world of computing simulation has experienced great progresses in recent years and requires more exigent multidisciplinary challenges to satisfy the new demands.

Due to the industrial maturity of one-purpose codes for the different fields, the production sector has increased its expectations, as they realize the need for solving in a more realistic way the problems they deal with, in order to stay competitive. Strong simplifications are the reason for which difficult problems could be solved in the past. These simplifications lead to a model as different from the real one, as the severity of the assumptions made. If we add every required accuracy in a concurrent world, then we need to relax the assumptions and become more general in the way of solving multidisciplinary problems.

The situation is not really new. What is novel is the need for solving multidisciplinary problems combining most of the information coming from different fields, obtaining a more precise information and therefore better optimization methods. This means solving more complex problems. There are many strategies to approach the solution of this kind of problems. A simple approach assumes that existing analysis methods are good and that people should create interfaces between already existing codes in order to solve multi-disciplinary problems. Unfortunately this strategy will simply not work for the monolithic solution of highly coupled problems and usually has

execution time overhead due to the data format conversion and memory overhead produced by duplicated data. These shortcomings provide a strong motivation for creating a multi-disciplinary programming framework which will allow dealing with these kind of problems in a more natural and flexible way.

It has been also planned to integrate another interesting topic in the multi-disciplinary solutions such as optimization. The vast amount of research works for the solution of industrial optimization problems constitutes a strong motivation to provide an affordable library as a numerical engine for optimization programs.

1.2 Problem

One of the relevant topics in the finite element method nowadays is the combination of different analysis (thermal, fluid dynamic, structural) with optimization methods, and adaptive meshing in one global software package with just one user interface and the possibility to extend the implemented solution to new types of problems, as an approach to a multi-disciplinary simulation environment.

Conventional finite element codes encounter several difficulties in dealing with multi-disciplinary problems involving the coupled solution of two or more different fields (i.e. fluid-structure, thermal-mechanical, thermal-fluid, etc.). Many of these codes are designed and implemented for solving a certain type of problems, generally involving a single field. For example to solve only structures, fluids, or electromagnetic problems. Extending these codes to deal with another field of analysis usually requires large amounts of modifications.

A common approach is to connect different solvers via a master program which implements the interaction algorithms and also transfers data from one solver to another. This approach has been used successfully in practice but has its own problems. First of all having completely separate programs results in many duplicated implementations, which causes an additional cost in code maintenance. In many cases the data transfer between different solvers is not trivial and, depending on the data structure of each program, may cause a redundant overhead of copying data from one data structure to another. Finally, this method can be used only for master and slave coupling and not for the monolithic coupling solution.

Some newer implementations have used the modern software engineering concepts in their design to make the program more extendible. They usually achieve the extendibility to new algorithms in their program but extending them to new fields is beyond their intent.

Using the object-oriented paradigm helps in improving the reusability of codes. This is considered to be a key point for streamlining the implementation of modules for solving new types of problems. Unfortunately many domain specific concepts in their design restricts their reusability for other modules.

The typical bottlenecks of existing codes for dealing with multi-disciplinary problems are:

- Predefined set of degrees of freedom per node.
- Data structure with fixed set of defined variables.
- Global list of variables for all entities.
- Domain based interfaces.
- IO restriction in reading new data and writing new results.
- Algorithm definition inside the code.

These shortcomings require extensive rewriting of the code in order to extend it to new fields.

Many programs have a predefined set of degrees of freedom per node. For example in a three dimensional structural program each node has six degrees of freedom $d_x, d_y, d_z, w_x, w_y, w_z$ where d is the nodal displacement and w the nodal rotation. Assuming all nodes to have just this set of degrees of freedom helps the developers to optimize their codes and also simplifies their implementation tasks. But this assumption prevents the extension of the code to another field with a different set of degrees of freedom per node.

Usually the data structure of programs is designed to hold certain variables and historical values. The main reasons for this design are: easier implementation, better performance of data structure and less effort in maintenance. In spite of these advantages using rigid data structures usually need important changes revision to hold new variables from another fields.

Another problem arises when the program's data structure is designed to hold the same set of data for all entities. In this case adding a nodal variable to the data structure implies adding this variable to all nodes in the model. In this implementation adding variables of one domain to data structure causes redundant spaces to be allocated for them in another domain. For example in a fluid-structure interaction problem, this design causes each structural node to have pressure values and all fluid nodes to have displacement values stored in memory. Even though this is not a restriction, it severely affects the memory efficiency of program.

Additionally, in single purpose programs, it is common to create domain specific interfaces in order to increase the code clarity. For example providing a `Conductivity` method for element's properties to get the element's conductivity as follow:

```
c = properties.Conductivity()
```

Though this enhances the code clarity, it is completely incompatible with extendibility to new fields.

IO is another bottleneck in extending the program to new fields. Each field has its sets of data and results, and a simple IO usually is unable to handle new set of data. This can cause significant implementation and maintenance costs that come from updating IO for each new problem to solve.

Finally introducing new algorithms to existing codes requires internal implementation. This causes closed programs to be nonextensible because there is no access to their source code. For open source programs, this requires the external developers learn about the internal structure of the code.

1.3 Objectives

The objective of this thesis is to design and implement a framework for building multi-disciplinary finite element programs. This framework is called Kratos and will help to build a wide variety of finite element programs from the simplest formulation, for example a heat conduction problem, to the most complex ones, like multi-disciplinary optimization techniques. From one side it will provide a complete set of flexible tools for fast implementation of experimental academic algorithms and from the other side it must be fast and efficient to be used for real industrial analysis.

Generality is the first objective in our design. A framework can be used if its generality is sufficient to fit a wide variety of finite element algorithms. The generality becomes more important when it comes to multi-disciplinary analysis in which, the code structure has to support the wide variety of algorithms involved in different areas.

Reusability is another objective in this design. Finite element methodology has several steps that are similar between different algorithms even for different types of problems. A good de-

sign and implementation can make all these steps reusable for all algorithms. This reduces the implementation effort and the maintenance cost of the code.

Another objective is providing a high level of extendibility for Kratos. The continuous innovation in finite element methodology may result in new algorithms with completely different requirements in the future. Thus supporting just a large number of current algorithms cannot guarantee the usefulness of the code in future. The solution is providing some ways to extend different parts of the code to new cases. For designing a multi-disciplinary code, extendibility plays an even more important role. Extendibility in a multi-disciplinary code is also necessary to support new problems in different fields. So extendibility is considered as an important objective for Kratos.

Good performance and memory efficiency are also objectives of Kratos. Solving multi-disciplinary industrial problems is the goal of Kratos and requires good performance and also good level of memory efficiency. Generality and flexibility are against the performance and efficiency of the code. So it is obvious that Kratos cannot achieve the same performance of a fully optimized single domain code. But the idea is to keep it as fast as possible and meanwhile increase the total performance in solving multi-disciplinary problems by reducing the data conversion and transfer between domains.

1.4 Solutions

Applying the object-oriented paradigm has shown to be very helpful in increasing the flexibility and reusability of codes. In this thesis, object-oriented design is successfully used to organize different parts of the code in a set of objects with clear interfaces. In this way, replacing one object with another is very easy, which increases the flexibility of the code, and reusing an object in some other places is also becomes more practical.

Design and implementation of a multi-disciplinary conceptual interface is another solution provided to previous problems. In the Kratos design, interfaces are defined in a very generic way and independent from any specific concept. The variable base interface resulting from this design is very general and solves the interface problems arising when extending the program to new fields.

A flexible and extendible data structure is another solution used to guarantee the extendibility of the code to new concepts. The proposed data structure is able to store any type of data associated to any concept. It also provides different ways for global organization of data required when dealing with multi-domains problems. The same strategy is applied to give flexibility in assigning any set of degrees of freedom to any node for solving new problems.

An interactive interface is provided in order to increase the flexibility of the code when implementing different algorithms. In this way a new algorithm can be introduced without the need to be implemented inside the program. This gives a high level of extendibility to the code and it is very useful for the implementation of optimization and multi-disciplinary interaction algorithms.

An automatic configurable IO module is added to these components providing the complete set of solutions necessary for dealing with multi-disciplinary problems. This IO module uses different component lists to adjust itself when reading and writing new concepts originating from different fields of analysis.

1.5 Organization

This thesis presents the design of a new object oriented finite element framework. The idea is to present a general view of its design and then divide the design procedure into its main parts while

describing each of them individually in separate chapters. Following this idea the layout of the thesis is organized as follows:

Chapter 2: Background In this chapter a background of finite element programming is given.

Chapter 3: Concepts It starts with a brief description of numerical analysis in general and continues by explaining the finite element method and its concepts with some detail. Finally it describes different design patterns used throughout this work and gives a brief description of some advanced C++ techniques used for writing high performance numerical applications.

Chapter 4: General Structure Explains the object oriented design of Kratos, its main objects, the code organization and also the separation between kernel and applications.

Chapter 5:Basic Tools Different tools provided to help developers in implementing their program are described in this chapter. Design of quadrature methods, linear solvers, and geometries are explained, and key points in their reusability and generality are mentioned.

Chapter 6:Variable Base Interface (VBI) Presents a new variable base interface to be used at different levels of the code. First, a motivation for designing the new interface is given. Then, this new interface is described and some applications are explained. After that there is a section about its implementation and the ways of designing interfaces using this method. Finally, some examples are given to show the capability of the VBI in practice.

Chapter 7:Data Structure This chapter first explains the basic concepts in programming containers and continues by describing different containers. Then presents new containers designed for finite element programming. Finally, a global scheme of data distribution in Kratos is provided.

Chapter 8:Finite Element Implementation Focuses on the components representing the finite element methodology implemented in Kratos. It describes the design of elements and conditions and also explains the structure of processes and strategies in Kratos. Finally it explains elemental expressions and formulations and their purpose of design.

Chapter 9:Input Output First it explains different approaches in designing IO modules and presents a flexible and generic IO structure. Then it continues with a part dedicated to interpreter writing, which consists of a small introduction to concepts and also a brief explanation on the use of related tools and libraries. Finally, it describes the use of Python as the interpreter with some description about the reasons for using Python, the interface library, and some examples to show its great abilities.

Chapter 10:Validation Examples It gives some examples of different finite element applications for multi-disciplinary problems implemented in Kratos. It also includes some benchmarks to compare the efficiency of Kratos with existing programs.

Chapter 11:Conclusions and Future Work Gives an overview of solutions and their effectiveness in solving multi-disciplinary problems and explains the future directions of this project.