Chapter 11

Conclusions and Future Work

11.1 Conclusions

Kratos, a framework for developing multi-disciplinary programs has been designed and implemented. It helps developers in implementing applications for different fields of analysis by providing input-output, data structures, solvers, basic tools, and standard algorithms. The applications implemented in this framework can be used for solving multi-disciplinary problems using any master and slave strategies or even by solving simultaneously. At this moment several solvers (incompressible fluid, structural, thermal, and electromagnetic) are implemented in Kratos. Combination of these applications are also used to solve different multi-disciplinary problems, specially fluid-structure interaction and thermal-structural problems.

This framework provides a high level of flexibility and generality which is required for dealing with multi-disciplinary problems. Developers in different areas can configure Kratos for their needs without altering the standard interface used to communicate with other fields in coupled analysis. Different applications like: the particle finite element method and explicit compressible fluid are implemented in Kratos which helped for validating its flexibility in handling different algorithms. Finally its python interface gives extra flexibility in handling nonstandard algorithms.

Several reusable components are provided to help developers allowing easier and faster implementation of their applications. Data structure, IO, linear solvers, geometries, quadrature tools, and different strategies are examples of these reusable components. Use of these components makes application development not only faster, but also ensures compatibility with other tools for solving multi-disciplinary problems.

Kratos is also very extensible at different levels of implementation. Each application can add its variables, degrees of freedom, `Properties`, `Elements`, `Conditions`, and solution algorithms to Kratos. The object-oriented structure and appropriate patterns used in its design make these extensions easy while reducing the need for modifications. The extensibility is also validated at all levels by implementing different applications varying from standard finite element applications to optimization procedures using Kratos and its applications.

Last but not least, the performance of Kratos is comparable even to single purpose programs and different benchmarks show this in practice. This makes Kratos a practical tool for solving industrial multi-disciplinary problem.

11.1.1 General Structure

An object-oriented structure has been designed to maximize the reusability and extensibility of the code. This structure is based on finite element methodology and many objects are designed to represent the basic finite element concepts. In this way the structure becomes easily understandable for developers with a finite element method background.

In this design `Vector`, `Matrix`, and `Quadrature` representing the basic numerical concepts. `Node`, `Element`, `Condition`, and `Dof` are defined directly form finite element concepts. `Model`, `Mesh`, and `Properties` are from the practical methodology used in finite element modeling complemented by `ModelPart`, and `SpatialContainer`, for organizing better all data necessary for analysis. `IO`, `LinearSolver`, `Process`, and `Strategy` represent the different steps of a finite element program flow. Finally `Kernel` and `Application` are defined for library management and its interface definition.

Kratos uses a multi-layer approach in its design which reduces the dependency between different parts of program. It helps in maintenance of the code and also helps developers in understanding the code. These layers are defined in a way such as each user has to work in the smallest number of layers as possible. In this way the amount of code that each users has to be familiar with is minimized and the chance of conflict between users of different categories is reduced. The implementation difficulties needed for each layer is also tuned for the knowledge of users working in it. For example the finite element layer uses only basic to average features of C++ programming but the main developer layer use advanced language features in order to provide desired performance.

11.1.2 Basic Tools

Different reusable tools have been implemented to help developers in writing their applications in Kratos. Several geometries and different quadrature methods are provided and their performances are optimized. Their flexible design and general interface make them suitable for use in different applications. Their optimized performance make them appropriate not only for academic applications but also for real industrial simulations.

An extensible structure for linear solvers has been designed and different common solvers have been implemented. In this design the solver encapsulates only the solving algorithms and all operations over vectors and matrices are encapsulated in space classes. In this way solvers become independent of the type of mathematical containers and can be used to solve completely different types of equations systems like symmetric, skyline, etc. This structure also allows highly optimized solvers (for just one type of matrices or vectors) to be implemented without any problem.

11.1.3 Variable Base Interface

A new variable base interface has been designed and implemented. All information about a concept or variable to be passed through this interface is encapsulated in the `Variable` class. The information about components of a variable also is encapsulated in the `VariableComponent` class which gives an additional flexibility to this interface. This interface is used at different levels of abstraction and proved to be very clear, flexible, and extensible.

`Variable` provides the type of data statically and objects can use it to configure their operations for a given type of data via template implementation. This type information also prevents the use of variables in procedures that cannot handle their type of data. Each variable has a unique key which can be used as the reference key in data structures. The name of variable as an string helps routines like IO to read and write them without requiring additional parameters. Finally it provides a zero value which can be used for initializing data independent of its type in generic algorithms.

Beside this information, variable provides different methods for raw memory manipulations. These methods are excellent tools for low level generic programming, specially for writing heterogeneous containers.

This interface has been used successfully in different parts of Kratos. Its flexibility and extendibility is demonstrated in practice and its evident contribution to readability of the code is shown. This interface played a great role in uniforming different concepts coming from different area of analysis.

11.1.4 Data Structure

New heterogeneous containers have been implemented in order to hold different types of data without any modifications. The `DataValueContainer` can be used to store variables of any type without even explicitly defining the list of them. This container is very flexible but uses a search mechanism to retrieve given variable's data. The `VariablesListContainer` only stores the variables defined in its variables list which can be have any type but its advantage is its fast indirection mechanism for finding the variables data. In Kratos these two containers are used alternatively in places where performance or flexibility are more important. Being able to store even the list of neighbor `Nodes` or `Elements` shows their flexibility in practice.

An entity base data structure has been developed in Kratos. This approach gives more freedom in partitioning the domain or in creating and removing `Nodes` and `Elements`, for example in adaptive meshing. Several levels of abstraction are provided to help users group model and data information in different ways. In Kratos the `Model` contains the whole model, divided to different `ModelParts`. Each model part can have different `Mesh`es which hold a complete set of entities in Kratos. These objects are effectively used for separating domain information or sending a single part to some process.

11.1.5 Finite Element Implementation

The `Element` and `Condition` classes are designed as the extension points of Kratos. Their generic interfaces provide all information necessary for calculating their local components and also are flexible enough for handling new arguments in the future.

Several processes and strategies has been developed to handle standard procedures in finite element programming. These components increase the reusability of the code and decrease the effort needed to implement new finite element application using Kratos.

Some experimental work has been done to handle elemental expression using a higher level of abstraction. In this way elemental expressions can be written in C++ but with a meta language very similar to mathematical notations and then can be compiled with the rest of the code using the C++ compiler. These expressions have been successfully tested and their performance is comparable to manually implemented codes.

Finally the `Formulation` is designed to handle nodal, edge based, or even elemental formulations in different forms of implementations. However these capabilities have not been implemented yet due to the lack of interest from developers.

11.1.6 Input-Output

A flexible and extensible IO module for finite element programs has been developed. It can handle new concepts very easily while Kratos automatically adds variables to its components and IO uses these components as its list of concepts. Therefore any application built with Kratos can use IO for reading and writing its own concepts without making any change to it. However, more effort is required to extend this IO system to handle new types of data.

This IO is multi-format. It can support new formats just by adding a new IO derived class an without changing any other part of IO. For example a binary format IO can be added using this feature.

An interpreter is also implemented to handle Kratos data files. Its format is relatively intuitive and similar to Python scripts. The major interpreting task is given to the Python interpreter. This flexible interpreter with its object-oriented high level language can be used to implement and execute new algorithms using Kratos. In this way the implementation and maintenance cost of a new sophisticated interpreter is eliminated.

11.2 Future Work

This work can be continued in different ways. First of all different extensions to the existing framework can increase the number of useful features provided by it. Besides these extensions, parallelization of the code is the next task to perform in order to guarantee its success in solving future large scale problems.

11.2.1 Extensions

Here is a list of suggested extensions to this work.

Basic Tools

New solvers and preconditioners should be added to extend the solving abilities of Kratos. Also a new type of linear solvers for very small system of equations should be implemented. They can be used for solving efficiently the small equations appearing in some algorithms like the patch recovery method [104].

Variable Base Interface

As mentioned before, there are some complexities related to incompatibility of variables and their components in this interface. This also is an open door for further improvements. A solution could be deriving the `VariableComponent` from `variable` and using some indexing mechanism to distinguish them. This can be completed by some traits to avoid virtual function calling in cases where a good performance is also needed. This is something to be implemented and tested carefully.

Finite Element Implementation

Creating new processes and strategies can increase the reusability of the code and also the completeness of Kratos. This can be done also by revising the processes and strategies implemented in different applications and adding a generic version of them to Kratos which could be usable for a wider set of applications.

As mentioned in section 8.5, an experimental elemental expression has been developed and tested. These part needs more components to be useable in a wide variety of formulations. Implementing missing components and practically use them can help the fast development of finite element formulations in Kratos, At the same time, it can be used to optimize the new formulations or even transform them automatically to parallel codes.

Formulations are another part of Kratos to explore. Adding nodal or edge based formulations to Kratos can be a good way to refine its design in practice.

Input-Output

Serialization is not implemented yet but is considered to be useful for automatization of problem loading and saving. Adding this feature to Kratos would help users to run longer problems and pause them whenever they want. Using an external library is considered to be a better solution than implementing it.

Supporting binary format for input can reduce significantly the data reading time. The multi-format feature of Kratos reduces a lot the effort necessary to implement it.

11.2.2 Parallelization

Beside the extensions mentioned before, parallelization of Kratos framework is the main task to be undertaken in the future. The growing size of problems and the increase of available parallel computing machines (even in the personal computers sector), stress the importance for parallelization of numerical codes. For this reason a big effort should be invested to parallelize Kratos for shared memory and distributed memory architectures.

Fortunately, several aspects of Kratos become useful in this process. Its entity based data structure makes the distribution of data over processors easier. Also, several layers of abstraction in the data structure will help the partitioning task which are needed for division of the model over processors. Finally the `Strategy` is designed to be parallelized with a very small effort.

11.3 Acknowledgments

This work is dedicated to Hengameh who left her family, friends and also her work to come to Barcelona with me. Her support and patience during these long years have been priceless.

I would like to thank Professor Eugenio Oñate for his support which made this project possible and to the whole Kratos team for their contributions and valuable feedback that was decisive for the evolution of Kratos.

I would also like to thank specially my good friend Dr. Riccardo Rossi for his great effort in developing Kratos as a very good engineer and programmer. He helped me to overcome several difficulties arising during this work not only with his practical solutions but also with his high motivation and belief in the project.

This work has been done in the memory of Francisco Javier Royo who started it but his short life did not let him finish it...