Requirements for a Solid Modeler Coupled to Finite-Element Mesh Generators

Ana Liddy Cenni de Castro Magalhães and Renato Cardoso Mesquita
Departamento de Engenharia Elétrica - Universidade Federal de Minas Gerais
Av. Antônio Carlos 6627, Pampulha, Belo Horizonte, MG, 31270-901, Brazil

Abstract—This paper presents the necessary requirements for a multi-representational solid modeler combining Constructive Solid Geometry (CSG) and Boundary Representation (B-rep) to be coupled to a 3D automatic finite-element mesh generator in electromagnetic problems. A modeler architecture and the functional specifications of the modules are defined. Implementation details, including object-oriented data structures and classes definitions are described.

Index terms—Geometric modeling, design automation software, finite element methods, software requirements, software design, data structures, object-oriented programming.

I - INTRODUCTION

The requirements analysis and software specification are very delicate tasks because they constitute the starting point for either success or failure. This is true especially in case of complex research systems, in which a detailed specification cannot normally be obtained at the first stages of the development. In the case of solid modelers construction for electromagnetism, this could not be different. Generally, solid modeling systems have basic components such as a data structure to store the representation schemes, a set of basic algorithms for handling these structures and user interface mechanisms that allow the implementation of different modeling and editing techniques. On Constructive Solid Geometry (CSG) based solid modelers, several requirements must be met to allow the construction of a valid object using primitives (such as spheres, blocks and cylinders) combined by means of Boolean set operations (union, intersection and subtraction). For the integration with Finite-Element Analysis, a Boundary Representation (B-rep), which describes an object in terms of its surfaces boundaries (vertices, edges and faces), must be generated for direct use in mesh generation. Moreover, the modeler should be specific to the problem at hand but also general enough to address future problems and requirements, avoiding redesign. This paper presents the necessary requirements for a multi-representational CSG and B-rep solid modeler coupled to a finite-element mesh generator for electromagnetic problems, under development at the Universidade Federal de Minas Gerais (UFMG), Brazil. Section II presents a brief functional specification of the modeler's basic components. Sections III, IV, V and VI specify the requirements and object-oriented data structures patterns related to these components, emphasizing the generality and reusability of the defined data structures. Section VII describes additional requirements related to parametrization and reusability.

Manuscript received November 3, 1997.
E-mails:{ana, renato}@cpdee.ufmg.br.
This work was supported in part by CAPES and CNPq, Brazil.

II - FUNCTIONAL STRUCTURE

A solid modeler should allow the creation, edition and access to a computational representation through interactive processes activated by the user. To achieve this purpose it is necessary to incorporate CSG description techniques [1], as well as visualization routines [2], allowing a better analysis of the geometric configuration for the model being designed. For electromagnetic simulation by finite-element method, automatic mesh generation routines based on a Boundary Representation [3] must also be coupled, so that application programs may use available model information to analyze it either statically or dynamically.

Generally, a solid modeler can be functionally composed by four main modules as shown in Fig. 1: users interact with the modeler to describe, edit and visualize solid objects through an interactive graphic interface that constitutes the Interface Subsystem. The received commands are translated into Modeling Subsystem requests, that perform modeling routines available in the system and are responsible for the execution of shape handling. In order to perform this handling, the Modeling Subsystem interacts with the Kernel Subsystem that retrieves and modifies the solid representations stored by the system. Once the model has been created, the user may request the finite-element mesh generation, which is performed by the Mesh Generation Subsystem. Thus, the generated model can be stored as a neutral file, including its geometric description, as well as its corresponding finite-element mesh, by means of which application programs and other interfaces may retrieve the information.

Fig. 1. Functional structure of the Modeler for electromagnetic simulation.
III - INTERFACE REQUIREMENTS

The Interface Subsystem is responsible for supplying the user with various ways of describing solid objects, as well as a set of shape handling operations that can be executed on these objects. The modeling tasks to be performed are described by the user through the interface, by selecting a sub-menu option, clicking at the icons available in the toolbars, or entering the command into the Command line, as illustrated in Fig. 2. The tasks are captured by the interface procedures that activate other subsystems procedures; the task results are received by the procedures and displayed by the interface, which must provide resources for visualization.

To guarantee the independence between the user interface objects, like buttons and menus, and the Modeler objects, it is desirable to turn the operation request itself into an object, which can be stored and passed around like other objects. This can be implemented using the command pattern structure described in Fig. 3 using OMT notation [4]: an abstract Command class declares an interface for executing operations; concrete Command subclasses defines a binding between a Receiver object and an action by storing the receiver as an instance variable and implementing execute operations to invoke the request; the Receiver has the knowledge required to carry out the request. Concrete Commands subclasses can also support undo and redo capabilities if they provide a way to reverse their execution. For commands that aren’t undoable and don’t require arguments, templates [5] can be used to avoid creating a Command subclass for every kind of action and receiver.

The user may create and edit objects interactively through the interface tools, in a suitable working plane, moving from a global coordinate system to a local one, chosen by the user. The interface must also provide mechanisms to avoid incorrect operation executions. Moreover, the Interface Subsystem must also provide resources for communication interfaces with other modelers and applications.

IV - MODELING REQUIREMENTS

The Modeling Subsystem is responsible for translating the objects’ description and operations obtained from the Interface Subsystem into commands for the creation or modification of the inner representations by the Kernel Subsystem. It involves primitive definition operations, Boolean operations, manipulation operations, visualizing operations, as well as geometric and non-geometric supporting operations. Moreover, the Modeling Subsystem must contain conversion procedures from CSG to Boundary Representation [6], both needing to be valid and stored. It must also answer geometric and topologic questions made by the system.

CSG models are typically represented in a CSG tree. Primitives form the leaves of the tree, while interior nodes correspond to either a Boolean set operation or a rigid transformation. To avoid solid definition redundancies, groupings (objects composed by primitives and operations) can be used to substitute primitives, defining CSG sub-trees. As the user treat primitives and CSG sub-trees identically, it is desirable that the code using these classes treats them uniformly, too. This can be implemented using a recursive composition pattern [4], a tree structure that represent part-whole hierarchies, as shown in Fig. 4. The key is the CSG Component abstract class, that represents both primitives and CSG sub-trees. This class declares operations like Draw, that are specific to primitives. It also declares operations that all CSG sub-trees share, such as operations for accessing and managing their children. Since primitives have no child, none of these subclasses implements child-related operations.

Aiming the integration with Finite-Element Analysis applications, the CSG representation must be converted into a Boundary Representation, which will be directly used in mesh generation. The algorithm that perform this conversion...
is called a “boundary evaluator”, and the resulting data structures record the faces of the object modeled. Typically this is implemented as a hierarchical data structure where faces are represented in terms of their bounding edges, and these in terms of their bounding vertices. Examples of such structures may be seen elsewhere [3],[6] - [8]. In addition to these basic types of objects and their relations, geometric information such as face and curve equations and vertex coordinates must be present.

Conventional boundary models can only represent manifolds objects (which requires that all edges separate exactly two faces and all vertices are surrounded by a single circuit of faces). This is a restriction for finite-element models where volumes with different materials touch each other, have faces that are shared by two objects and edges that are shared by more than two faces. A representation of the internal structure of the parts, one for each connected piece of material, is unattractive, because high level algorithms such as Boolean set cannot be supported easily [7]. Moreover, identical surface mesh in both sides of the part must be guaranteed so that the 3D mesh is compatible.

So-called non-manifold models have been introduced to deal with these issues and also to include lower dimensional entities than solids in a solid model data structure [6] - [8]. They are constructed from similar geometric elements as classical boundary representations, and the basic topological elements are vertex, edge, loop, face, shell, region and model. The relationship between these elements of the hierarchy is much like the one depicted in Fig. 5 [6]. For any element within the hierarchy, there is a direct path to the element which is one level higher and also to the element which is one level lower. For the elements vertex, edge, loop and face, there is a distinction between the existence of the element and instances of the use of the element. This allows multiple topological elements to share the same underlying form and geometry. Moreover, each topological element makes reference to a separate geometric element, which enables the evolution to richer geometric forms while continuing to share a common set of topological elements with a stable interface.

V - KERNEL REQUIREMENTS

The Kernel Subsystem is responsible for the management and access to the data structure: any access to object representations required by other subsystems is made through this manager, that constitutes the modeler’s kernel. Thus, the inner representation information can be read by procedures of other subsystems, but must be created and modified only by the Kernel procedures.

The most used data structure in a multi-representational CSG and B-rep solid modeler are lists and trees. It is interesting to have a way to access their elements without exposing its internal structure. Moreover, it is desirable to traverse these structures in different ways, depending on what is to be done. It is possible to take the responsibility for access and traversal out of the structure, by implementing an iterator [4],[5] object, as shown in Fig. 6. The iterator and the structure are coupled. The iterator class defines an interface for accessing the elements, keeping track of the current element in each structure: it knows which elements have already been traversed. Separating the traversal mechanism from the structure let different iterators be defined for different traversals, without enumerating them in the structure interface. An Abstract Structure class that provides a common interface for manipulating structures is defined, as well as an abstract Iterator class that defines a common iteration interface. Then, concrete Iterator subclasses for the different structure implementations can be defined. As a result, the iteration mechanism becomes independent of the concrete structure classes and support polymorphic iteration.

Different data types are manipulated as components in lists and trees in the Kernel Subsystem. Aiming to avoid code duplication for these components, common behavior should be factored and localized in a common class providing adaptable storage containers, which implement the invariant parts of an algorithm once and leave it up to subclasses to implement the behavior that can vary. This can be implemented through the template pattern [4], which defines a skeleton of an algorithm in an operation, deferring some steps to subclasses, which redefine them without changing the algorithm’s structure. Types produced as template class instances can be used anywhere ordinary types can be used, and template functions can be used to define generic routines. As shown in Fig. 7, a template method defines an algorithm in terms of abstract operations that subclasses override to provide concrete behavior. By defining some of the steps of an algorithm using abstract operations, the template method fixes their ordering, but it lets its subclasses vary those steps to suit their needs. However, designers must be alert to minimize the number of operations that must be overridden.

Fig. 5. Example of data structure for non-manifold models

Fig. 6. The iterator pattern provides a uniform interface for traversing different structures
The Kernel is also responsible for providing a communication interface with other modelers and other applications, as well as supplying mechanisms for object descriptions, storage in a permanent data base.

The generation of a neutral file containing standard formats for data transfer is important for the data interchange. Acting as a pre-processor, the solid modeler must organize its geometric information in a neutral file, so that a processor can retrieve and treat this information; the processing results can also be put in the neutral file so that the information can be treated by a post-processor. A standard file format, compatible, for instance, with STEP [9],[10], should be used.

VI - MESH GENERATION REQUIREMENTS

Mesh generation methods for 3D models normally create the mesh from the topology and geometry of the boundary representation. All data required by the surface and volume meshing algorithms are obtained from the non-manifold B-rep data structure.

The majority of the mesh generation methods are based either on tetrahedrization algorithms for a point set or on the partition of the boundary representation in elements. As explained in [10], the meshing process begins with the subdivision on the edges surrounding the faces. Biases may be assigned towards one or both the endpoints of the edge, allowing a refined mesh where necessary. On the next stage of meshing process, triangular meshes should be generated on all faces, projecting the faces into a 2D parameter space, meshing this projection with a 2D meshing algorithm and reprojecting the mesh to its original space, taking in account distortions between these two spaces [10]. The last stage is the generation of the tetrahedral mesh by a Delaunay method.

The superficial mesh generation from the model’s boundary demands attention. Firstly, there must be compatibility between the boundary’s vertices and the superficial mesh nodes, that is to say, the nodes that are on the edge shared by boundary faces of the object must belong to these faces and include the boundary model’s vertices. Furthermore, there must be a mechanism that allows the specification of the generated superficial mesh density, so that regions that need a more refined mesh can be obtained and controlled by the user.

VII - ADDITIONAL REQUIREMENTS

Solid modelers are very useful for the design and representation of specific, final products. However, if the user aim is the design of an optimal shape, his interest will concentrate on prototype generation from various format specifications, aiming at a later shape adjust through appropriated tools. For such systems, some requirements need to be achieved, such as: availability of new facilities for the conceptual project and better tools for the model’s construction and edition; the possibility of reusing previous projects; the possibility of changing the size and location of all the elements and/or project geometric components. Parametric modelers [7] allow the user to define geometric expressions instead of specifying the object with concrete dimensions. A constrained-based parametric modeling, emphasizing the constructive approach is a good solution for this requirement [11]. The possibility of reutilization is another requirement that should be considered. If a new model that is being constructed is similar to a previous one, it is desirable that the model can be reused.

VIII - CONCLUSION

This paper presented an informal specification of a solid modeler coupled to an automatic finite-element mesh generator under development at UFMG. The functionality, architecture, components and main desirable requirements were described and discussed. Implementation details, including object oriented data structures and classes definitions were also presented, emphasizing the use of patterns, leading to the design of a specific model to the problem at hand but also general enough to address future problems and requirements, minimizing redesign.

REFERENCES