Engineering with Computers (2002) 18: 339–351

Engineering with Computers

Attribute Management System for Engineering Analysis

Robert M. O'Bara,[1] Mark W. Beall[1] and Mark S. Shephard[2]

[1]Simmetrix Inc.; [2]Rensselaer Design Research Center, Rensselaer Polytechnic Institute, Troy. NY, USA

Abstract. *This paper presents the design and implementation of an attribute management system that supports the specification of information, past that of the domain definition, needed to qualify an engineering analysis. The information managed by this system includes various order tensors needed to specify the analysis attributes of material properties, loads, and boundary conditions as well as additional data constructs used by the analysis such as strings, and references to either other attributes or model entities. The system supports general dependencies and variations of this attribute information as well as its association with the various geometric entities which constitute the geometric domain being analyzed. In addition, since the information is coupled with the model entities themselves, the system can be used to store information needed to control the discretization process of the geometric domain. Since the information can be both spatially and temporally varying, an expression subsystem was also designed into the system. The framework was designed using object-oriented techniques, implemented in C++, and can be easily maintained and extended.*

Keywords. Attribute management

1. Introduction

The application of Computer-Aided Design (CAD) technology has become commonplace in industry. Although the product information in CAD models could be employed by Computer-Aided Engineering (CAE) systems, the impact to date of CAE has fallen far short of its potential. The application of today's numerical analysis procedures by design engineers can be effective only if they are automated and if the quantitative results produced reliably predict the parameters requested.

To achieve this, it is necessary to utilize automatic mesh generation [1–4] that interacts directly with the geometric representation of the domain, robust finite element analysis procedures, error estimation, and adaptivity. The automatic mesh generation procedures can be integrated with finite element analysis procedures which can determine approximate values to the desired solution parameters. The solutions obtained then need to be examined to determine if the results obtained are of the requested level of accuracy. Substantial progress has been made on methods to estimate the mesh discretization errors [5,6], while methods to estimate analysis modeling errors are beginning to receive increased attention. If the error estimates indicate that the model and/or its discretization needs improvement, the model construction and/or mesh generation procedures can be used to perform the required adaptations before the next analysis step.

The application of an automated adaptive analysis environment of this type requires procedures to support the geometry-based specification of the loads, material properties, boundary conditions, and initial conditions required to complete the specification of the physical problem to be simulated. Historically, *ad hoc* methods have been used for the specification of analysis attributes in which each attribute was defined directly in terms of the numerical analysis discretization. Such specifications can not improve the approximation to the attribute when the mesh is modified as dictated by an adaptive analysis procedure. The correct, and more natural, specification of the analysis attributes is to associate them directly with the geometric representation of the domain to be analyzed. Assuming the classification information of the mesh with respect to its geometric model[1] is already maintained, the amount of memory needed to associate the attributes can be drastically reduced. An additional benefit of associating attributes directly with the model is that information related to controlling the discretization process itself can be represented in the system. Starting from a high level statement of requirements for such an attribute

[1] Mesh classification refers to the unique association of each mesh topological entity with the geometric model topological entity that if lies on or within [9].

specification capability [7], this paper presents an attribute management system for the geometry-based definition of tensorial information that supports general variations and dependencies.

Section 2 overviews the functional requirements of an analysis attribute manager to support: (i) the organization of individual analysis attributes into the correct collection for a particular analysis (Sect. 3); (ii) the information that defines analysis attributes (Sect. 4); (iii) the specification of general distributions for attributes and dependencies among the attributes (Sect. 5); and (iv) the association of the geometry-based attributes with the domain definition housed in a CAD system (Sect. 6). Section 7 discusses some of the implementation issues.

2. Requirements for Analysis Attribute Specification

2.1. Analysis Attribute Information

An examination of the properties of analysis attributes indicates those that qualify the physical problem are tensorial quantities, and therefore require a general scheme to specify various order tensors [8]. Tensorial quantities must be defined with respect to a coordinate system and may include various forms of symmetry which must be taken into account for efficiency. Therefore, the components of the structure used to define tensor attributes are (i) the order of the tensor, (ii) the coordinate system the attribute is defined in, (iii) the symmetries possessed, and (iv) the dependence of the tensor on other quantities.

Tensors are specified using values based on a specific coordinate system; however, by maintaining the coordinate system information, the tensor information can be transformed into any requested coordinate system. The system provides the following capabilities for handling tensor information.

- Creating tensors of any order.
- Defining tensors in a coordinate system with given values of its components. These values can either be numerical or expressions.
- Defining the symmetry properties of the tensor. This allows tensors to be created that are defined not on a component by component basis but on the consequences of the symmetries. For example, the fourth order tensor of a linear isotropic elastic material is based on only two independent parameters.
- Evaluating the tensor in a different coordinate system.
- Changing a tensor's defining coordinate system via a transformation.
- Including tensors as part of an expression which can depend on position, time, and other tensors. A tensor may contain expressions that also introduce other dependences.

In addition to tensors, other common forms of attribute information used in numerical simulations include integers, reals, and character strings. Another type is a reference to a model entity (or entities) that can be used to define other information. For example, the loading on one surface may be related to the minimum distance relative to another surface represented as a model entity attribute. Another example of using model references is in the case of describing the properties of a fan in a CFD analysis. One component of the fan's properties is the identification of the fan's inlet and outlet surfaces which would be represented as references to the model's surfaces. Similarly the system provides a mechanism to model information by referencing other attributes in the system.

2.2. Data Expressions

An attribute may be a function of space, time, or some other attribute. In addition the system supports attributes that receive their definition from a variety of other sources, such as:

- The results of a previous analysis where the spatial description is given in terms of quantities defined over some discretized version of the geometry (e.g. applying a temperature from a thermal analysis as an attribute for a stress analysis).
- A function which depends on geometric operators (e.g. a traction may depend upon the distance between two model entities).

In order to represent non-constant information the system supports mathematical expressions that consist of operators, constants, and user defined variables. A variable can be locally defined inside the attribute or inherited when the attribute information is associated with the geometric model.

The functions supported for the mathematical expressions include:

- Arithmetic: + − */X^Y (including vector and matrix versions).
- Trigometric: sin() cos() tan() arcsin() arccos() arctan().
- Nesting of functions (e.g. (), {}, etc.).

- Interpolations: Linear, Bi-Linear, Tri-Linear, Spline-based, etc.
- Integration and Differentiation.
- Conditional.
- User-defined Functions.

In the case of modeling expressions, modeling operators such as closest point, intersection, union, and subtraction are also included.

The expression sub-system also allows users to easily add new operators and functions in order to address future requirements.

2.3. Association Information

Analysis attributes are associated with a specific portion of the domain being analyzed. In the problem definition case where analysis attributes are primarily associated with the specification of boundary value problems, the association of the attributes with the various topological entities in the boundary representation of the geometric model is a natural choice.

To ensure the ability to represent attributes specified on all possible domains, a full non-manifold topology similar to that described in References [10,11] is needed (Fig. 1). The mesh generated for the model is also defined by the appropriate set of base topological entities (vertex, edge, face, and region) and the classification of the mesh entities to the geometric model is maintained [9]. This mesh/model representation is key to being able to apply the geometry-based attribute specification methods described below.

In addition to the original geometric entities in the model, auxiliary geometric information, which aids in attribute specification, may be required to fully capture the analyst's intentions. Common examples of auxiliary geometries are scribed edges on a model face used to indicate that a traction is only applied to a portion of a face, or projection geometry that aids in the specification of an attribute, such as a wind load distribution on a vertical plane. To properly reflect the association of such attributes the geometric model requires augmentation to account for the auxiliary geometry. For example, the curve used to denote the end of a traction on a face must be used to split the face during augmentation.

2.4. Organization of Attributes

To support the effective specification of attributes for the complete set of related analyses, while at the same time making it efficient to collect the attributes required for each specific analysis, an organizational structure is needed for the purpose of describing sets of attributes. The organizational structure must effectively support a design process for scenarios where multiple physical behaviors must be evaluated. In many cases, the result of one analysis represents part of the problem definition of another. For example, consider the situation of performing thermal, electrical, and thermal-mechanical analyses of an electrical component. Though the three analyses are quite different, there is an overlap of attribute information. The base materials are the same for all three analysis types, while the boundary conditions and loading conditions vary among the three. The thermal analysis would study various thermal load distributions. The thermal-mechanical analysis would use the resulting temperature fields as input to its load cases. The ability to effectively organize hierarchies of attributes as needed for each of these analyses is critical to a useful attribute management system.

To meet these needs the attribute manager must be able to structure information:

- Consisting of several pieces of specific information which must remain together in order for a specific analysis to be well defined. This concept is represented in the system as a Case.
- To support relationships between information components. For example, a load on a given model entity may be represented by the vector expression {a, 0, 0}, while another load on a different entity is defined by {0, b, 0}. Using

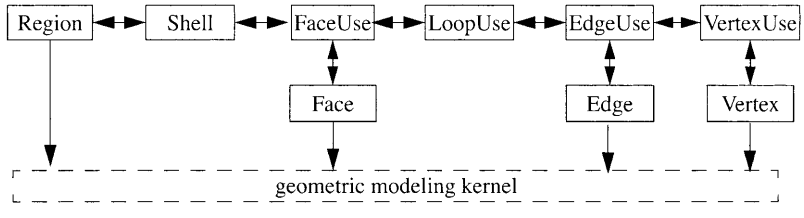

Fig. 1. Model topological adjacency information and relation to model geometry.

hierarchical information, one can impose a relationship between the variables a and b such that a = 2b.

An analysis can typically be partitioned further into three different categories of information, each of which can be represented as a Case in the system:

- *Problem Definition Case* – attributes defining the physical problem being modeled such as boundary conditions, loads, and material properties.
- *Discretization Case* – attributes defining the discretization required for the analysis including target mesh sizes, anisotropic meshing characteristics (such as boundary layers), and mesh matching constraints.
- *Solution Strategy Case* – attributes that define the numerical procedures used to solve the problem. This would include the type of solver to be used, convergence criteria, and the polynomial order.

By partitioning an analysis' information into these three cases, the user can better reuse the information in defining different analyses which share common aspects. For example, two analyses might share the same solution strategy and problem definitions but require different discretizations.

3. Organization of Attributes

An appropriate model must be defined for the structuring of the information to be controlled by the attribute manager. The Analysis Information Model (AIM) defined here represents a set of analysis attributes that can share common information and are thus built upon each other. The AIM is represented as a directed acyclic graph that consists of analysis information structuring nodes (AISNs) that are used to represent the analysis attribute information. There are three basic classes of AISN:

- *Information Node* – used to represent conceptual attribute specification information.
- *Group Node* – used to represent hierarchical information.
- *Case Node* – used to represent a complete description for a specific analysis[2]. Cases also store the associations of the information with geometric models.

[2] A 'complete description' is a conceptualization. The system will not check the information stored under the case for completeness since it does not know what information is needed. Verification of a complete specification requires an understanding of the problem domain.

All AISNs can define variables internal to the node itself. These variables can be constants or expressions. During the attribute association process, the variables of a node are passed down to its children and can be used to model data dependences.

3.1. Analysis Information Structuring Nodes

3.1.1. Information Nodes (IN).
Information Nodes (IN) represent the conceptual types of information introduced in Sect. 2.1. They are specialized to contain specific types of information. For example, there are String INs, First Order Tensor INs, Real INs, etc. INs behave similarly to variables in that they can be either constant or contain an expression. For example,

```
// tempnode is a 0th order tensor and
load 1 node is a 1st order tensor
load 1 node[0] = "2 *sin($t)";//Define
the first component of the load to be a
function of time
tempnode = 10.0;// Define the temperature
to be a constant value
```

As with all AISNs, information nodes can have other nodes as children in order to represent more complex information. For example, to properly model density, the node itself would be modeled as a zero order tensor. In addition, if the node were to represent density using a Boussinesq approximation, then it would have the following children nodes:

- *Expansivity* – represented as a zero order tensor.
- *Reference Temperature* – represented as a zero order tensor.

3.1.2. Group Nodes.
The Group Nodes are used for structuring the information and defining interdependencies among the information nodes. Group Nodes are also useful in clustering information that should be applied to the same model entity.

3.1.3. Case Nodes.
A Case Node represents a point in the graph that completely defines the information for a particular analysis. A Case Node's type can be used to indicate the problem domain being addressed by that subgraph. It can be used by a validity checking to verify that the information is complete. A Case Node can reference all types of nodes including other cases. In this way one case can be built on

top of another, resulting in hierarchical structuring of cases.

In addition the Case Node also stores both a reference to the model being associated as well as the associations with the model entities themselves. By storing the model reference in a Case Node, the designer is forced to think in terms of what information is being applied to a particular model. More than one model can be incorporated in a case by referencing other case nodes. A case node can also re-define the associations that other cases have made thereby increasing the reusability of existing analysis definitions.

3.2. Attribute Specification

The graph of AISNs represents the analysis information that is independent of the geometric model. The attribute management system also provides a mechanism to relate this information back to the geometric model. The AISNs are combined with links to the model entities to construct the attributes. The system supports association of the attributes to the appropriate mesh entities within the analysis.

3.2.1. Model Entities.
The model entities represent the topological components of a geometric model. The model entities that the AISNs can be associated with are the vertices, edges, faces, and regions of the model. For certain types of analyses it is also useful to be able to associate the AISNs with the uses of specific model entities. For example, consider an analysis in which there are two material regions in contact with a lubricant of infinitesimal thickness between them. Assuming a representation (at the model and analysis level) where the lubricant is given a coefficient but no thickness, the following attributes need to be specified:

- The lubricant coefficient is assigned to the model face.
- The surface roughness parameter for the material on one side is associated with the face use on that side, while the surface roughness parameter for the material on the other side is associated with the face use on that side.

3.2.2. Analysis Structuring Information Nodes vs. Attribute Objects.
The system makes a clear distinction between AISNs in the graph and the attribute objects that are assigned to the model topology for a specific case. One could think of an AISN as an attribute generator [12]. During the association of a case, an AISN, which is contained in one of the case's association paths, generates attribute objects that are connected to the topology. These attribute objects are what we typically think of as the attributes of a model entity. A single AISN can generate multiple attribute objects due to the following:

- The AISN is included in multiple association paths.
- The AISN is being applied to more than one model entity via its distributivity property.

In the example shown in Fig. 2, AISN1 is being applied to the face of the rectangle. In addition, the node has its distributivity property set to INHERIT (indicated by the grayed node) which means all lower order topological entities associated with the face will also have the information associated with it (indicated by attribute object a1). The 'ar' attribute objects reference the explicit attribute a1.

3.2.3. Types of Attribute Objects.
There are two main types of attribute objects:

- *Associated Attributes* – these are created from AISN's that have been explicitly associated with model entities. By explicitly associated we mean that it is the highest level Information Node whose path to a case node is contained in an association path.
- *Component Attributes* – these are created from Information Nodes whose ancestors have created Associated Attributes.

Note that neither Case or Group Nodes directly create attributes and therefore are not seen by the parts of the analysis that deal with querying attribute information directly from model entities. The reason we make distinctions as to how an attribute object is associated with a model entity is to be able to query the attributes based on the association. For example, consider the previous Density definition using a Boussinesq approximation. If a model region has Density associated with it, then querying the region for Density will return the appropriate attri-

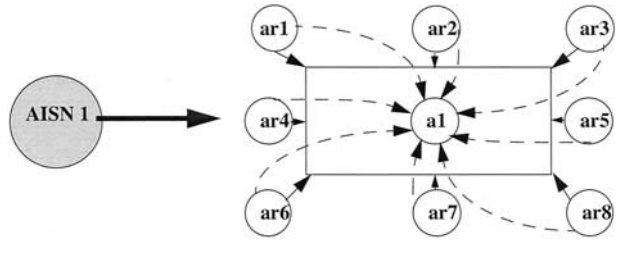

Fig. 2. Example of attribute inheritance.

bute while at the same time asking for Expansivity would not return any attribute since the density's expansivity is not directly associated with the region but with the density's attribute itself.

4. Analysis Attribute Information

The data that actually defines an analysis attribute is stored in the AISNs. The specific information recognizes the requirements of the analysis and how the numerical analysis procedures would want to query and evaluate it.

4.1. Analysis Information Structuring Nodes

An AISN consists of the following information for each attribute defined:

- *Name* – not necessarily unique.
- *InfoType* – represents the type of information being represented.
- *ImageClass* – can be used to represent a 'sub-class' of a specific InfoType.
- *Defined variables*.
- *Unique Persistent Identifier* defined by the system.

The name and InfoType of the AISN are defined by the application. For example a density attribute could have the name 'density 1' and InfoType 'Density'. In most situations it is the InfoType that is more important to the analysis since it indicates what type of information the attribute represents. Since the name of an AISN does not have to be unique, the system has to provide a mechanism that will allow the node to be uniquely referenced through out its lifetime.

In addition to the InfoType, the application can also add an ImageClass name to the AISN. One common use of the ImageClass is to indicate a 'sub-class' of the InfoType. In the previous density example, the application may assign 'Boussinesq' as the ImageClass name indicating that the attribute will have the additional attribute components of 'Expansivity' and 'Reference Temperature'.

4.2. Expressions and Variables

An AISN can be defined as a set of variables which are managed by the node. The types of variables include:

- Reals.
- Integers.
- Tensors.
- Strings.
- References to AISNs
- References to Model Entities

In addition the system provides predefined variables for time ($t), space ($x, $y, $z) and the parametric space of model edge and face entities ($u, $v). The system also defines a special variable $me that is a reference to the model entity associated with the attribute itself. Since information may be varying in both time and space, an application may need to specify both spatial and time coordinates when evaluating a variable. For example.

```
Variable a = aisn.defineVar ("a", REAL,
DEFINE);// Declare a to be real
a = "sin($t) *$x;// a is both spatially
and temporally varying
Variable   b   =   aisn.defineVar("b",
INTEGER, DEFAULT) // Declare b to be an
integer
b = 10; // b is a constant;
```

In addition, a definition may be used to define a default value that could be overridden by an ancestor AISN. This allows for greater reusability of AISNs and for information to be parametrized. In the above example, b is defined to be a default which means any ancestor node of the AISN can override b's value. Consider the example shown in Fig. 3. Since a was defined with 'DEFINE', under either case the value of aisn's 'a' would be 10.0 while b, which was defined with 'DEFAULT' would have the value of 2 when processing Case 1 and would be time varying when processing Case 2.

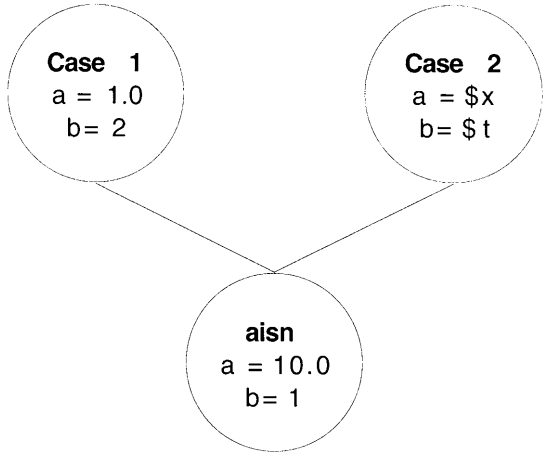

Fig. 3. Example of defining variables.

4.3. Mathematical Expressions

The expression sub-system allows an application to specify an infix expression as a character string and internally parses it into a stack of variables and functional objects that represent the expression in Polish Notation [13]. The justification for using a stack based approach versus an expression tree was to facilitate adding new functions and operators. In tree-based approaches each node representing an operation or function has n-arcs coming into it where each are represents an augment being passed in. Therefore, the structure of a functional node depends on its signature. By using a stack-based representation, all objects in the system have the exact same evaluation signature which is to simply pass in a result stack that the object manipulates. While forming the expression stack, sub-expressions involving constants are simplified if possible. For example the expression string '4 *cos(PI/3.0)' would be simplified to 2, while '$t *cos(PI/3.0)' would have the following expression stack:

```
$t
0.5
add-function
```

In the former, the expression is evaluated only once, while in the latter, the expression needs to be re-evaluated as needed. In addition, the expression system is strongly typed and allows for both functional and operator overloading. For example the dyadic '+' operator can be defined for both numerical summation and string concatenation.

5. Modeling Dependencies

The attribute system provides mechanisms to model two basic forms of dependencies:

- Distributivity – how information is inherited based on model topology.
- Data Constraints and Parametrizations.

5.1. Distributivity

Distributivity of an AISN refers to how the node's information relates to the model entity and its closure. For example, information applied to a face may also need to be associated with the edges and vertices of the face. Forcing the user or application to do this explicitly would be tedious and prone to errors. This can be especially useful when assigning information to the entire model. The system's design includes the following distributivity types:

- *None* – information is explicitly associated with model entities.
- *Inheritable* – in addition to the topological entity explicitly associated with the information, lower order topological entities that are associated with the model entity implicitly have the information associated with them.
- *Closure* – the information is applicable to the closure of the model entity but not the entity itself. In this context closure refers to the topological entities that are one order lower than that of the model entity itself.

The distributivity type Closure behaves similarly to Inheritable but restricts the type of entities on the model entity's boundary that will inherit the information. For example, the user may wish to assign a temperature attribute to all the faces of a model region. By using the Closure option, this can be easily achieved. In addition, if the model is later altered and the set of faces of the region is changed, this mechanism still insures that all the faces will be properly assigned a temperature value.

5.2. Data Constraints and Parameterizations

In addition to topological dependences, the information inside a node itself may be related to another node's information. Consider the case of representing two velocity constraints in which the user requires the magnitudes to be the same. This relationship can be modeled using variables in defining the magnitudes of the constraints and then defining the variable in a common ancestor node (see Fig. 4). In addition to modeling dependencies between information stored in different nodes, variables also allows the information to be structured parametrically. In the example shown in Fig. 4, the magnitudes have been parametrized with respect to Case 1's variable 'm' allowing a series of analyses

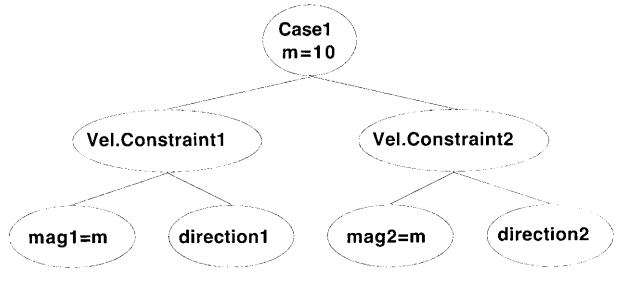

Fig. 4. Velocity Constraint Example showing the use of variables.

to be defined simply by changing the variable's value.

6. Associating AISNs with Geometric Models

This section describes the process of creating attribute objects by associating the AISNs under a Case Node and the appropriate geometric model entities.

6.1. Specifying Model Associations

The notation used to represent an association between the graph of AISNs and model entities is

{model entities}:{AISNs}

The left side of the association represents the list of model entities that will be assigned any attributes generated by the association. If the list is empty then the attributes are attached to the Case Node that owns the association. One application of using an empty model entity list is in dealing with the majority of the information pertaining to non-physical attributes like the Solution Strategy Case. Information related to solvers and time integrators are not typically associated with any particular model entities but to the case itself. The right side of the association represents paths in the graph that contain the list of AISNs. This may contain at most one Information Node which would be the node creating the corresponding attributes. In the situation where the list does not contain any INs, the association refers to all IN's that fit the following criteria:

- The path from the Case Node to the Information Node contains the list of AISNs in the association.
- The Information Node does not have another Information Node as a parent in the graph, i.e. the Information Node is not a component of another Information Node.

Consider the example shown in Fig. 5, which depicts the loading of a model consisting of a single region for an elasticity analysis. In this example, there are three loads defined. Two of the loads are identical and are applied to two different faces. In addition, a third load is applied to one of the already loaded faces. Finally, an attribute describing the material properties of the entire model is applied to the region itself. The benefits of using paths stored in case nodes (instead of directly assigning modeling information to the information nodes) is reusability and flexibility such as the situation where another Case Node wants to reuse the Material Property and assign that information to a different model entity (or even a different model). In addition, the use of Group 1 in the association insures that f1 will be properly loaded. In the future, if there are additional boundary conditions required for f1 then they can be simply added to Group 1. No additional model association information would be needed.

In the situation where a Case Node references another Case Node, the application can redefine the associations defined by the referenced Case Node. This is shown in Fig. 6, which defines Case 2 by reusing Case 1. In this example, the new analysis case requires the loading that was applied to f1 to now be applied to f3. In addition, a new load is now applied to f4. By allowing the reuse of nodes and model associations, the common requirements between related cases can be explicitly captured as demonstrated by these two scenarios.

Figure 7 shows an example of reusing a problem specification on a completely different model. The original specification loaded a simple cylinder which could be used as simple test case. The second case represents the same loading but uses a more complex model. Note that once again some of the information has been parameterized thus facilitating changes to the magnitudes of the load and the constraint.

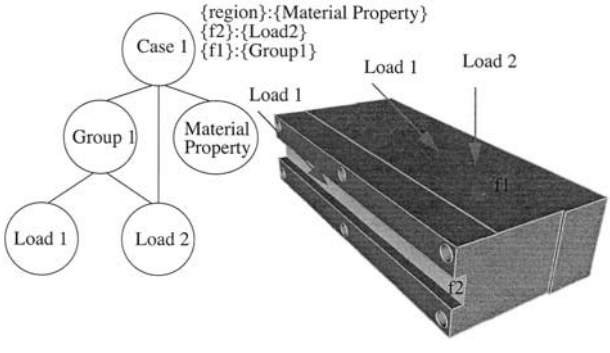

Fig. 5. Association of attributes with model entities.

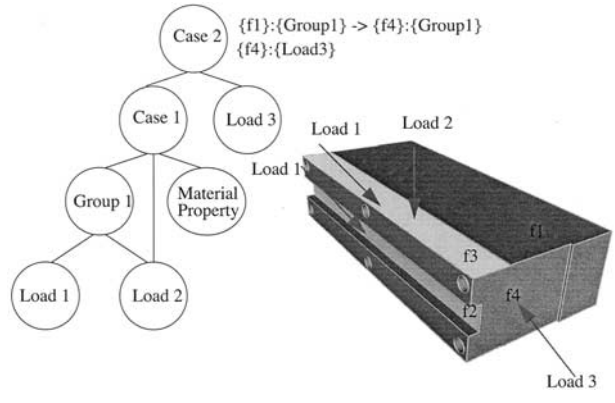

Fig. 6. Redefinition of associations.

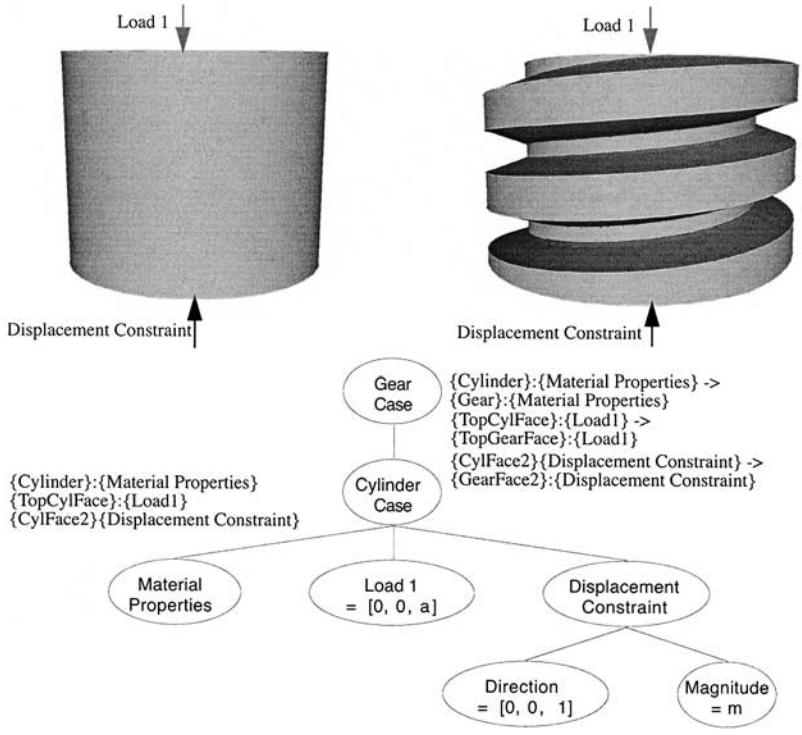

Fig. 7. Example showing reuse of an analysis definition using a completely different model.

6.2. Query Methods Based on a Model Entity (me)

Once the attributes have been created via the association process, the system provides the application query functions to retrieve the attributes directly from a model entity (or Case Nodes in the situation of model associations that do not specify any model entities). The available queries include:

- me.attributes() – return all attributes associated with the model entity me.
- me.attributes(InfoType) – return all attributes of a specific InfoType.
- me.attribute(InfoType) – return the first attribute of a specific InfoType.

6.3. Query Methods Based on an Attribute Object (ao)

Once the analysis has retrieved an attribute object, it can then query information such as the attribute's InfoType, provide access to its children attributes in the scenario of its Information Node having component nodes, and evaluate the attribute at a specific time and/or spatial location. The attribute object itself provides access both to its Information Node and to the model entity or Case Node that it is associated with. The available functions include:

- ao.entity() – return the model entity associated with the attribute object.
- ao.parent() – return the parent attribute.
- ao.node() – return the Information Node of the attribute.
- ao.children() – return all the component attributes.
- ao.childByType(InfoType) – return the first component attribute of a specific InfoType.
- ao.eval(t) – Evaluate the attribute at time = t.
- ao.eval(p) – Evaluate the attribute at position p.
- ao.eval(t,p) – Evaluate the attribute at time = t and position p.
- ao.isConstant() – returns true if the attribute is a constant value.
- ao.isTemporallyConstant() – returns true if the attribute is time independent.
- ao.isSpatiallyConstant() – returns true if the attribute is spatially constant.

6.4. Auxiliary Geometry

In many cases, auxiliary geometry has to be created in order to apply the information in a node to the geometric model. For example, consider the task of

loading face f1 of the bottom block due to the presence of the top block depicted in Fig. 8. To properly load the face, the f1 needs to be split into two pieces that represent the portions of the f1 that are in contact and not in contact with the top block. The main issue is how to automatically re-define the existing model association information with the geometry during the creation of the auxiliary geometry as well as associate the specific information to the newly created topology. The first issue of maintaining the original relationships is solved by creating an auxiliary model entity that represents the original entities that were affected by the modeling operation. In the previous example a new entity is created that contains the two newly created faces. The entity's boundary is the five edges that made up the original face's edge loop. The model associations are then updated by replacing f1 with the auxiliary entity.

In addressing the second issue of referring to model entities created by the modeling operation, each operation defines sets of model entities that the operation creates. These sets can then be used in defining model associations. For example, consider the biphasic analysis of cartilage tissue that forms part of a human knee as depicted in Fig. 9. In this scenario, only the portions of the tissues that are in contact need to be loaded. In addition, since the majority of fluid flow occurs along the contact boundary, the mesh should be refined along these 'edges'. To identify these contact areas, an imprinting operation is used to define new edges and faces on the original tissue surface with respect

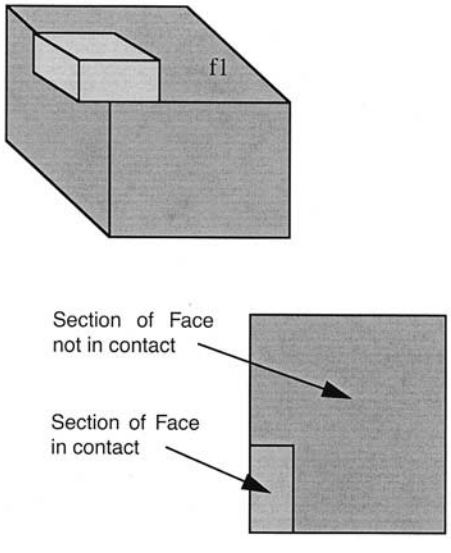

Fig. 8. An example of modifying original model topology to properly define attributes.

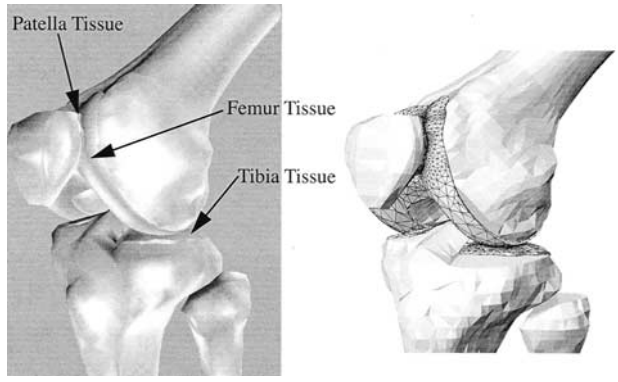

Fig. 9. Using auxiliary geometry to define mesh refinement (geometry courtesy of the Orthropedics Research Lab of Columbia University).

to the other 'tool' tissue surface. The imprint operation defined the following sets of modeling entities:

- Set of faces that are in contact with the 'tool' {Fc}.
- Set of faces that are not in contact {Fn}.
- Set of edges that form the boundary of the contact areas {Ec}.
- Set of edges that are not in contact {En}.

Figure 10 shows the attribute specifications required for the femur cartilage with respect to the patella cartilage. In addition to imprinting the contact areas and specifying meshing and material properties, the loading information, which is defined as a bilinear interpolation using proximity data acquired from actual measurements in a file call p.data, is associated with the contacting faces.

7. Implementation Issues

The attribute management system was designed based on classic object-oriented patterns and data

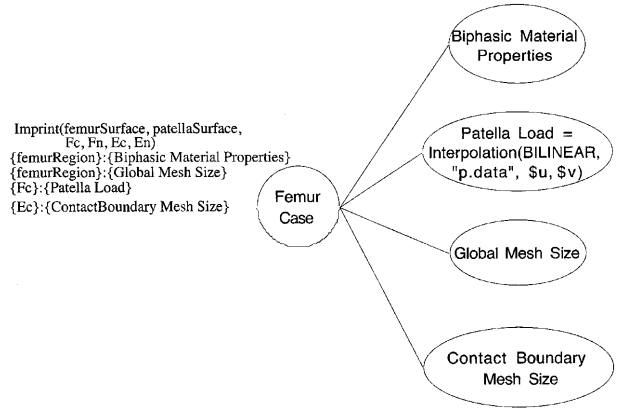

Fig. 10. Analysis information for the femur tissues based on the patella.

management concepts. The system was implemented as a set of extendable classes using the C++ language. The classes represent specific types of AISNs and attribute objects that can be defined in the system, variables and expressions that can be declared and referenced in nodes and attributes, and the manager object itself which is responsible for creating and deleting AISNs as well as saving and retrieving the information in terms of file I/O.

7.1. Interfaces and Tools

The attribute system provides both C and C++ interfaces for application development. The interfaces provide the ability to define and manage the AISNs as well as the model association information. In addition, the interfaces allow an application to associate a Case Node and thereby create attributes and attach them to their corresponding model entities or Case Nodes. Once the attributes have been defined, the application can query and evaluate the attributes themselves. The application also has access to the expression system and can specify and assign expressions to Information Nodes and defined variables. Finally, the application is provided with file I/O related functions that allow the AISN graph to be saved to and retrieved from a file.

In addition to providing function interfaces, attribute information editors have also been designed. Figure 11 shows an example of one editor that was implemented. To facilitate defining attribute information, an attribute definition procedure was developed which allows a user to register both the InfoType and ImageClass of an Information Node and then describe the required and/or optional children Information Nodes. The editor then can read these definitions at run time and present the appropriate dialog windows to the end user.

7.2. The Association Process

For the attribute system to be extended by adding new types of AISNs to meet future needs, the association process used to create and attach attribute objects to model entities must be formalized. The association process for a basic AISN is defined by three processes:

- Based on an inherited traversal state, define a new state based on the node's information.
- Apply the new traversal state.
- Restore the original traversal state.

The traversal state is defined by the following information.

- The current traversal paths of AISNs.
- List of model associations that have not been matched.
- List of model entities requiring attributes.
- List of Case Nodes requiring attributes.
- List of Attributes requiring children attributes.
- List of defined variables.
- The model currently being used.

The basic process of creating a new traversal state performs the following operations:

1. Add the node to the path.
2. Determine which model associations match the new path and remove those associations from the list.
3. Add the model entities and/or cases from the match associations to the appropriate lists.

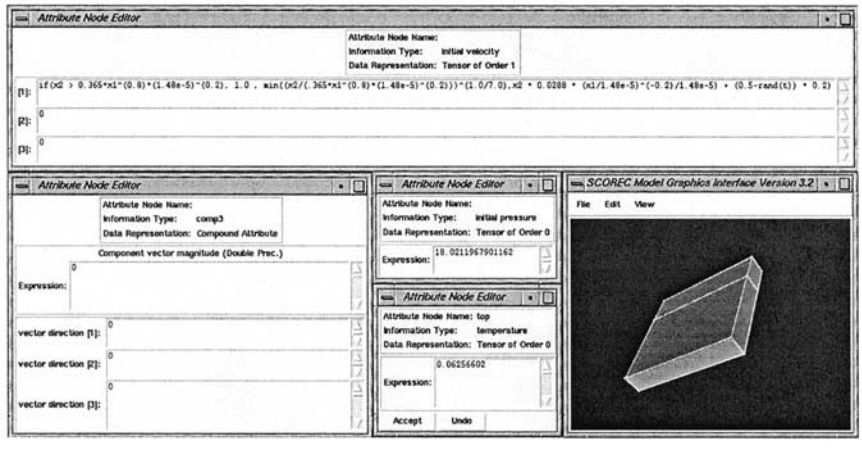

Fig. 11. An example of an attribute editor based on the attribute system.

4. Add the node's variables to the variable list. DEFINED variables are inserted at the beginning and the DEFAULT variables are appended to the end.

In the process of applying the traversal state, Case and Group Nodes, simply pass the new state to their children nodes' associate process. In the case of Information Nodes, applying the traversal state performs the following steps:

1. If the traversal state's lists of Attributes, Cases, and model entities are empty there is nothing to apply and the process just returns.
2. If the traversal state's lists of Attributes is empty then create the appropriate attributes and attach them to the corresponding model entities and/or Case Nodes.
3. If the list of Attributes is not empty then create the appropriate attributes and attach them to the Attributes in the list.
4. Create a new traversal state based on the current one, with the Case and model entity list set to be empty and the Attribute list defined as the set of attributes created by the Information Node.
5. Associate all the node's children using the new state.

Classes derived from the Information Node class need only override the attribute creation function. By designing the association process in this way, new classes of Information Nodes can be easily added to the system. When an attribute is created the traversal state's list of variables is used to locate the definitions of variables referenced in the attribute's expression (if it has one). Note that, since default variable definitions are always appended to the end of the list, these definitions are used only if there are no over-riding definitions of the same name.

All nodes with the exception of Case Nodes use the basic association process as defined above. Case Nodes add their model associations to the traversal state and set the state's model to be that of the Case Node being associated (if there is one). The basic association process is then invoked. Afterwards the model associations of the Case Node are removed and the state's model is restored.

7.3. Assigning One Model Entity per Attribute

The system is designed to create unique attribute objects for each association with a model entity. This is to prevent the need for passing the model entity back to an attribute object during evaluation. Consider the case where an information node is distributed to several model entities. If, during association, the node created only one attribute object and assigned it to all of the model entities, there would be ambiguity when evaluating the attribute object. Which model entity is asking for the information?

There are two approaches to solving the problem of information distribution:

- Creating and assigning attribute objects to each topological entity that is in the scope of distribution.
- When inquiring an attribute, the search is not restricted to the model entity but also to all higher level entities (this is assuming that either inherited or closure distribution is requested). The search continues until either the attribute (with the correct distribution mode) is found or the search reaches the model level.

The first method is more memory-intensive, since this means creating multiple attribute objects. These objects must be as lightweight as possible to be effective. The second method requires more processing power to perform the inquiry. The system is designed to use the first approach since we feel that execution time is a more limited resource. To improve efficiency, information assigned to the entire model is not to be replicated but stored in a special look-up list.

8. Closing Remarks

This paper has presented a generalized approach and associated capabilities for the specification and control of the attribute information needed to support reliable engineering analyses in an integrated design environment.

Since all analysis information is associated with the highest level domain definition, in this case a non-manifold solid model, it is easy to properly associate the loads, material properties, and boundary conditions with any spatial discretization of the domain generated for the application of a numerical analysis. This supports the ability to use any selected discretization approach and these analyses can be applied automatically given an appropriate algorithm to create the domain discretization. This approach also supports the introduction of analysis reliability through the application of adaptive analysis technologies that change the discretization as the analysis proceeds.

The structures used to support the specification and grouping of analysis attributes are also consistent with the needs of an industrial design process in which databases of various information such as material properties and environmental conditions (to specify boundary conditions) are used through the design/manufacture process. They also effectively support the application of multiphysics analyses in which information from one engineering analysis provides input information for another analysis process.

References

1. George, P. L. (1991) Automatic Mesh Generation. Wiley
2. Shephard, M. S., Weatherill, N. P. (1991) Int. J. Num. Meth. Eng. 32(4)
3. Shephard, M. S., Georges, M.K. (1992) Reliability of automatic 3-D mesh generation. Comp. Meth. Appl. Mech. and Engng. 101:443–462
4. Shephard, M. S. (2000) Meshing environment for geometry-based analysis. Int. J. Numer. Meth. Engng. 47(1–3):169–190
5. Clark, K., Flaherty, J. E., Shephard, M.S. (1994) Applied Numerical Mathematics. North Holland, The Netherlands, 14(1–3)
6. Oden, J. T., Demkowicz, L. (1992) Computer Methods in Applied Mechanics and Eng., Special issue on the reliability of finite element computations. North Holland, 101
7. Shephard, M. S. (1988) The specification of physical attribute information for engineering analysis. Engineering with Computers 4:145–155
8. Beju, I., Soos, E., Teodorescu P. P. (1983) Euclidean Tensor Calculus with Applications. Abacus Press
9. Beall, M. W., Shephard, M. S. (1997) A general topology-based mesh data structure. Int. J. Numer. Meth. Engng. 40(9):1573–1596
10. Gursoz, E. L., Choi, Y., Prinz, F. B. (1990) Vertex-based representation of Non-manifold boundaries. Geometric Modeling Product Engineering, North Holland, Amsterdam, pp. 107–130
11. Weiler, K. (1988) The radial edge structure: a topological representation for non-manifold geometric modeling. Geometric Modeling for CAD Applications, 3–36
12. Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995) Design Patterns, Elements of Reusable Object-Oriented Software. Addison Wesley
13. Sedgewick, R. (1992) Algorithms in C++. Addison Wesley