Chapter 4. Model Objects

Topic: *Entity, *Model Object

An *entity* is the most basic ACIS object. It is implemented in the C++ class ENTITY. All entities have a common set of functionality, such as the ability to save and restore themselves to and from a file, copy themselves, and debug themselves. All other geometric and higher level ACIS model objects are derived from the ENTITY class.

A *model object* is any object that can be saved to and restored from a save file (.sat or .sab). ACIS model objects are implemented in C++ using a hierarchy of classes that are derived from the ENTITY class.

This chapter describes the ENTITY class and how the model geometry classes, model topology classes, and attribute classes (all derived from class ENTITY) are integrated together to form a model. For information about specific classes, refer to the reference templates in online help.

The ENTITY Class

Topic: *Entity, *Model Object

The C++ class ENTITY provides basic model management functionality for all permanent objects in an ACIS model. The principle components of the model derived from ENTITY are topology, geometry, and attributes. Entities are always created on the heap and never on the stack. The C++ new and delete operators are redefined and overloaded.

All entities have a common set of functionality, such as the ability to be saved and restored to and from an ACIS save file, copy support, and debug support. This functionality is defined in two ways:

ENTITY	This class defines data and methods that all children of ENTITY *inherit*.
ENTITY_FUNCTIONS	This macro is included by all children of ENTITY, and provides a *local instantiation* (as opposed to inheritance) of much of the remaining functionality that is common to all model objects.

Although the ENTITY class itself does not represent objects within the modeler, any class that represents a permanent model object in ACIS must be derived from ENTITY in order to inherit the common data and functionality that is mandatory in all permanent objects.

ACIS provides run-time mechanisms for identifying the type of any object derived from ENTITY, as well as its level of derivation. Refer to the *3D ACIS Application Development Manual* for information about identifying model objects.

Figure 4-1 shows the derivation of some principal classes from ENTITY to illustrate the relationships. This diagram does not list all classes, nor even all classes in the principal components that are derived from ENTITY. All of the individual classes are described in detail in reference templates in online help.

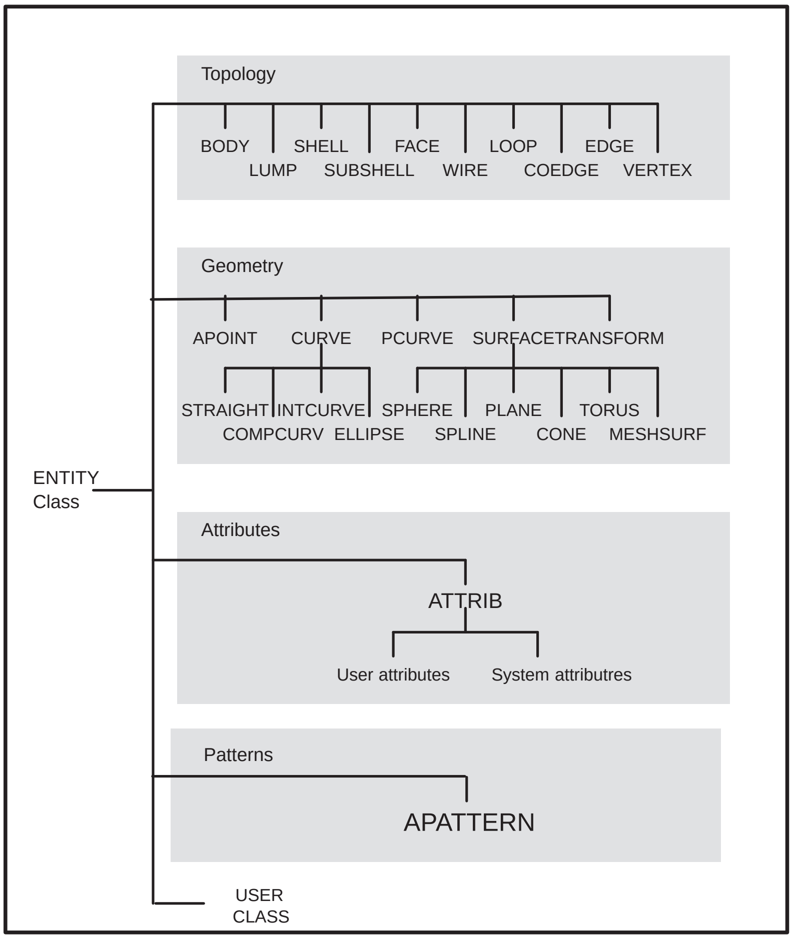

Figure 4-1. Model Object Class Derivations

Model Object Class Relationships

Topic: *Entity, *Model Object

ACIS brings together the separate worlds of wireframes, surfaces, and solids by allowing these alternative representations to coexist in its data structure. Wireframe entities can coexist with solids and sheets, and can share edges, coedges, and vertexes. From this coexistence comes the ability to define mixed dimensionality models and a variety of nonclosed models, such as a plane with three bounding edges and one unbounded (infinite) direction.

ACIS delimits objects with a *bounding box* that is used to determine intersections efficiently. If the bounding box of two objects intersect, the objects are likely to intersect, so more precise intersection calculations are performed. If the bounding boxes do not intersect, the objects cannot intersect, so the more precise intersection calculations are not necessary.

ACIS decomposes solid, sheet, wire, and mixed bodies into the classes shown in Figure 4-2. These classes provide the data and methods necessary for a true boundary representation solid modeler. Pointers up and down the hierarchy allow quick traversal of the data to determine such things as whether two entities share an edge or a vertex. As Figure 4-2 illustrates, the topology classes contain pointers to the corresponding geometry classes.

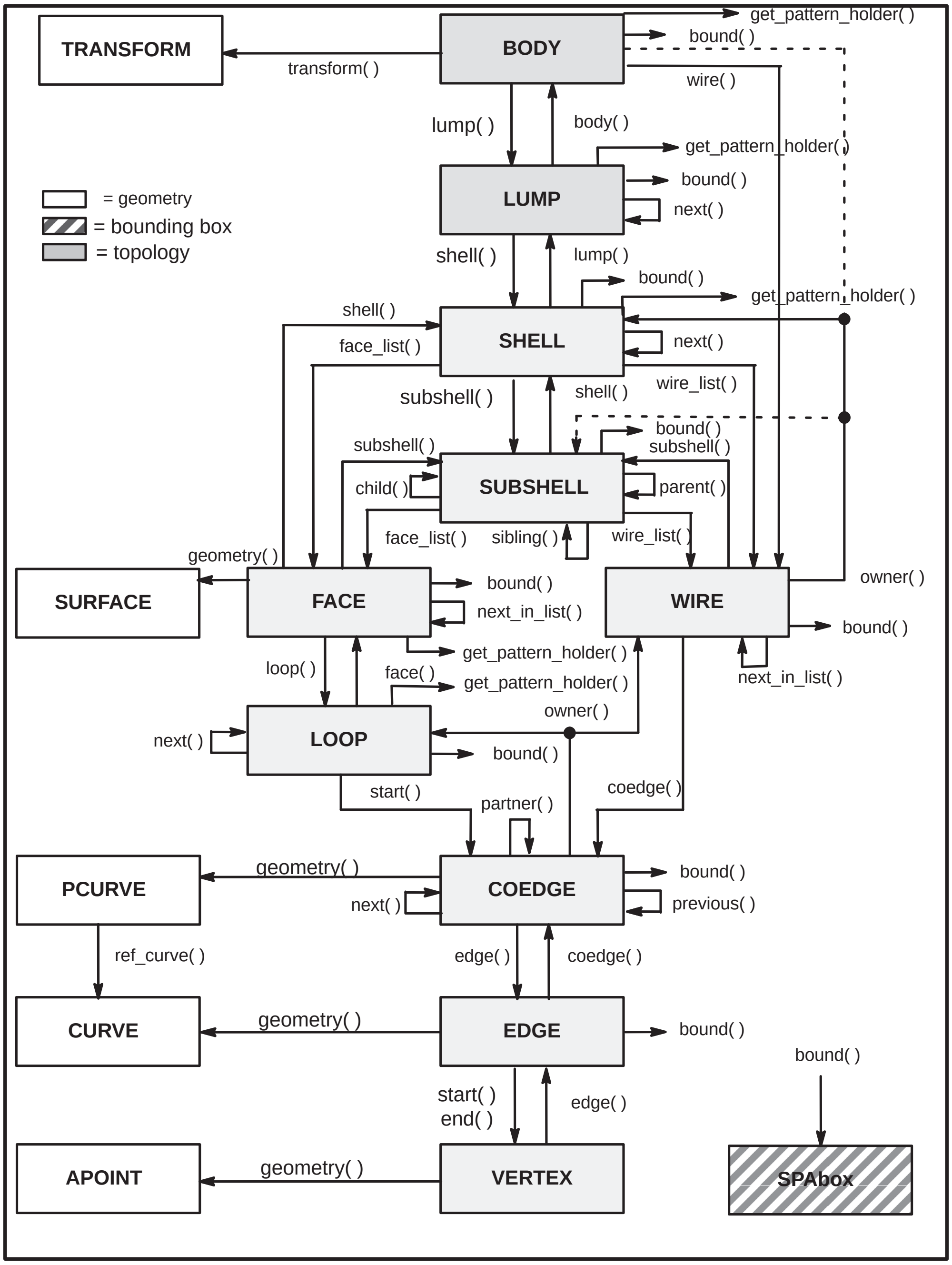

Figure 4-2. Implementation of Model Objects