

SEDRIS Data Model

Richard H. James
Orion Development Group, Inc.

1. Introduction

1.1 Purpose

One of the major objectives of SEDRIS is to provide a mechanism for unambiguous environmental data interchanges. To accomplish this objective, SEDRIS developed a data model encompassing all the data requirements of synthetic environments used in every type of simulation application. The data contained in the SEDRIS data model covers every environmental domain – terrain, ocean, atmosphere and near-space. Prior to SEDRIS, projects attempting to provide an environmental data interchange mechanism focused on development of a format for the interchange medium. Unfortunately, a format-focused approach led to ambiguous data since the underlying meaning and relationships of the data cannot be captured with just a description of the data's format. The SEDRIS data model provides not only a clear description of the data but also defines the relationships between the data – relationships which are critical to ensuring a correct interpretation by the users.

1.2 References

1. Transcript of Farid Mamaghani's Spring '96 Presentation
2. Transcript of Paul Birkel's Spring '96 Presentation
3. FAQuest Transcript, STRICOM, 23 - 24 July 1997
4. Interview with Bill Horan, 4 November 1997

2. What is a Data Model ?

A data model is a graphics-based design tool used to provide an identification of the types of data, with their attributes and relationships, that are used within a system. The use of a data model is a software engineering technique for unambiguously articulating information about the data. A data model provides a capability to articulate what kinds of things (*i.e.*, data) are contained in the system. A data model is not an implementation of a database; rather, it is a depiction of the types of data within the system as well as a justification of why the data is needed. A data model is supported by a data dictionary which provides additional textual (*i.e.*, non-graphical) information describing each type of data including its attributes and relationships.

As with any design tool, a data model uses a specific notation to provide a shorthand method to depict the information required to describe the data with their attributes and relationships. This notation allows the data model to be presented in a clear, graphical manner while still providing a means to fully capture the design of the data. The data model is a

representational scheme not a format – although a data model can be easily mapped to a format. It is far more difficult to map a format to a data model.

An important benefit of a data model is that it provides a means to ensure all the system's data types are captured along with a complete and unambiguous definition of their attributes as well as their relationships with other data types in the system. Since a data model is a notational depiction, it can easily be examined, challenged, and improved to ensure that it is a complete representation of the system's data.

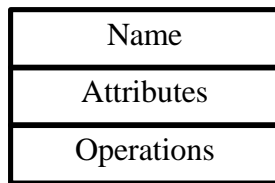
The data model identifies classes of data as well as the primitive data types. The data classes depict the organizational structure of the data. The primitive data types define the fundamental data elements that are used to define the other data classes. The identification of data classes also provides the capability to describe the relationships between the classes. All of this information – classes, primitives, and relationships – can be thought of as a specialized language with its own grammar. Given that we have a language, we can now minimize any differences of interpretation of the data contained in the data model.

3. What is the SEDRIS Data Model Notation ?

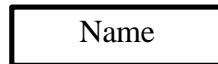
The SEDRIS Data Model is based on Rumbaugh's Object Model Template (OMT) notation. The SEDRIS Data Model uses some minor extensions to the OMT (*i.e.*, extra notations) but, for the most part, the notation is OMT. The OMT notation follows the object-oriented methodology for organizing the data. Although development of any subsequent database system does not have to also use object-oriented techniques, use of the object-oriented methodology during the analysis and design phases is very beneficial in database applications due to its data centric paradigm.

Central to the OMT notation is the concept of a class of data. A class is an abstract, user-defined description of a data type. It identifies the attributes of the data type and the operations that can be performed on instances (*i.e.*, objects) of the data type. By using abstractions of data type classes, we can describe the design of a software system using the same terminology as the corresponding real-world objects and their features. Recognize that in SEDRIS, the Data Model contains the data classes for the *representation* of the natural environment, not abstractions of the objects in the natural environment. The SEDRIS Data Model does not contain a data class for a "tree" or a "tank" but instead contains the data classes that are used to *represent* a "tree" or a "tank" or any other kind of object found in the natural environment. This concept of modeling representational data types is further explained in the following section.

A class of data has a name, a set of attributes that describe its state, and a set of operations that can be performed on the attributes. In OMT notation, a class is represented as a box:



To minimize the amount of information on a diagram of the SEDRIS Data Model, we chose to put all attribute information in the Data Dictionary. Since the data types defined by the SEDRIS Data Model are used to capture the contents of an environmental database, they do not experience dynamic changes. Therefore, no operations are defined for any of the SEDRIS data types. So in the SEDRIS Data Model, a class is represented simply as a rectangle with a Name:

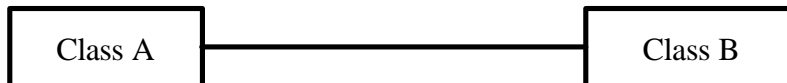


There are two kinds of classes represented in the SEDRIS Data Model: abstract and concrete. Abstract classes never have instantiations; in other words, no objects of this class type will be created. Concrete classes can have objects created from them. Abstract classes are used to define the common attributes that may be used by any of its subordinate, or child, classes. The relationship between parent and child classes (called an inheritance relationship) will be described more fully in a later paragraph in this section. To denote abstract classes in the SEDRIS Data Model, the class rectangle is shaded:



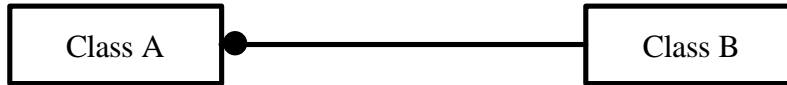
Shading rectangles to denote abstract classes is a deviation from the official OMT notation.

As stated earlier, a critical type of information to be captured in a data model is the relationships between the classes of data. There are three kinds of relationships between classes of data types: association, inheritance, and aggregation. Each relationship type has a different notation. The weakest kind of relationship is association, shown as a simple direct line between two classes:



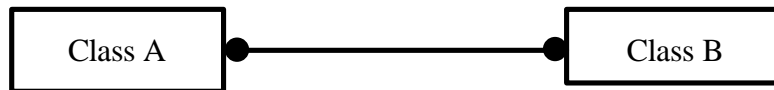
This notation indicates that every Class A is associated with one Class B, and every Class B is associated with one Class A.

Symbols at the end of the relationship lines are used to show multiplicity of the classes. No symbols, as used in the diagram above, indicate exactly one of each class type. A filled circle means zero or *more* classes at that end of the relationship:

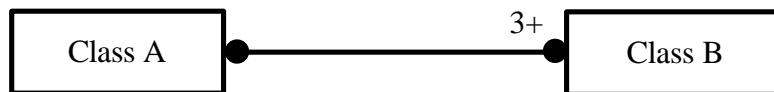


This notation says that each Class A class is associated with exactly one Class B class, but each Class B class is associated with zero or more Class A classes. In other words, a Class A class must have an associated Class B class, but a Class B class does not have to have an associated Class A class.

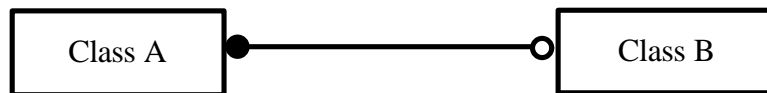
The following diagram says that Class A can have, but is not required to have, an association with Class B; and, Class B may or may not have an association with Class A:



If a number with a plus sign (+) is shown at either end of a relationship line with a closed circle, it means that instead of zero or more classes, there are at *least* the number indicated or more classes. The following diagram illustrates that Class A is associated with at least 3 or more Class B classes. Class B remains associated with zero or more Class A classes.

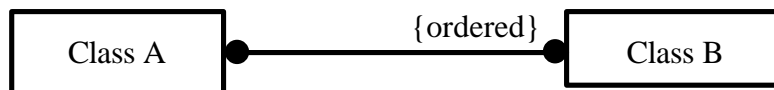


An open circle at either end of the relationship line indicates zero or *one* class:



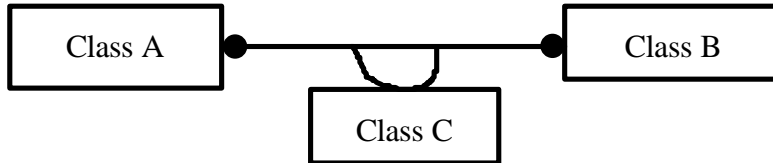
The above notation says that Class A is associated with either none or only one Class B class, while Class B is associated with zero or more Class A classes.

To indicate if the order of the classes is important to the relationship, an {ordered} tag is shown on the appropriate end of the relationship line:



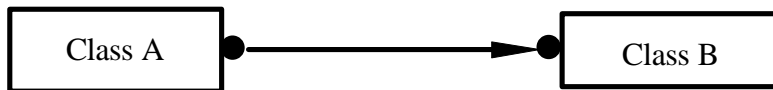
This notation indicates that Class A is associated with zero or many Class B classes and if there are any Class B classes, the order is critical. Unless the end of a relationship line has the {ordered} tag, the collection of classes at that end are considered to be unordered.

The association relationship between two data classes may have its own attributes. To show this, an association class is drawn with a “horse collar” attachment:

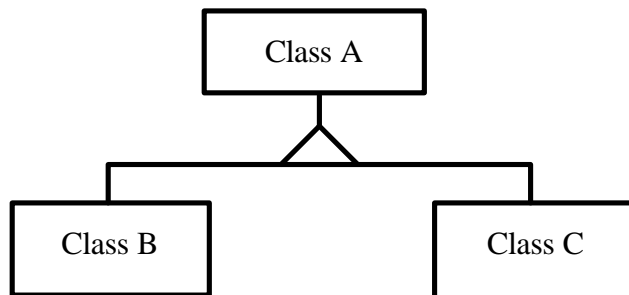


This notation indicates that each Class A is associated with zero or more Class B classes and each Class B is associated with zero or more Class A classes. In addition, it shows that for any (Class A, Class B) relationship pair, the relationship is also associated with exactly one Class C class.

The last notation used with association relationships is the arrowhead to depict one-way associations. This notation is used to indicate that when traversing the data model the relationship can be followed only in the direction of the arrowhead. In the following diagram, Class A is associated with zero or more Class B classes while each Class B is associated with zero or more Class A classes. However, when traversing through the data model, Class A “knows” that it is associated with Class B, while Class B does not “know” that it is associated with Class A. A traversal can go from Class A to Class B, but not from Class B to Class A.

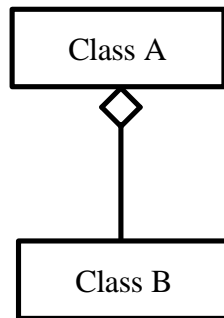


The next type of relationship between two classes is the inheritance relationship, introduced earlier. One of the classes in an inheritance relationship is the Parent or Super Class while the other class (or classes) is the Child or Subordinate Class. In this type of relationship, the Child Classes “inherit” the attributes of their Parent Class, while having unique, additional attributes of their own. This is sometimes referred to as the “is-a” relationship – the Child Class “is-a” type of the Parent Class. The inheritance relationship is depicted in the data model with a triangle:

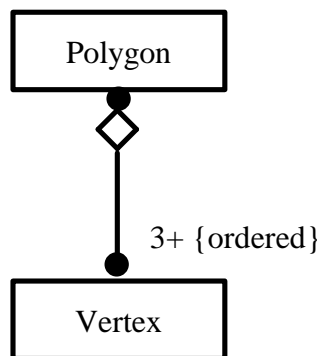


This diagram shows that Class B and Class C are both child classes of Class A. This means that both Classes B and C have the common attributes inherited from Class A and also have unique attributes of their own. If Classes B and C did not have unique attributes that distinguished between them, then they would really be the same class. Sometimes the parent class (in this example, Class A) will be an abstract class (cannot be instantiated) while the child classes will be concrete classes (can instantiate or create objects of this type). There is no multiplicity symbol used for the inheritance relationship since the SEDRIS Data Model does not support multiple inheritance. The “is-a” relationship is always one to one. Not allowing multiple inheritance relationships is a deviation from the official OMT notation.

The third and final type of relationship that can be shown between data classes in the SEDRIS Data Model is the aggregation type. This relationship is sometimes referred to as the “has-a” relationship. In this relationship, Class A is an aggregation of (or contains) a Class B class. In other words, Class A “has-a” Class B. This relationship is denoted by a diamond positioned at the aggregate end of the relationship line:

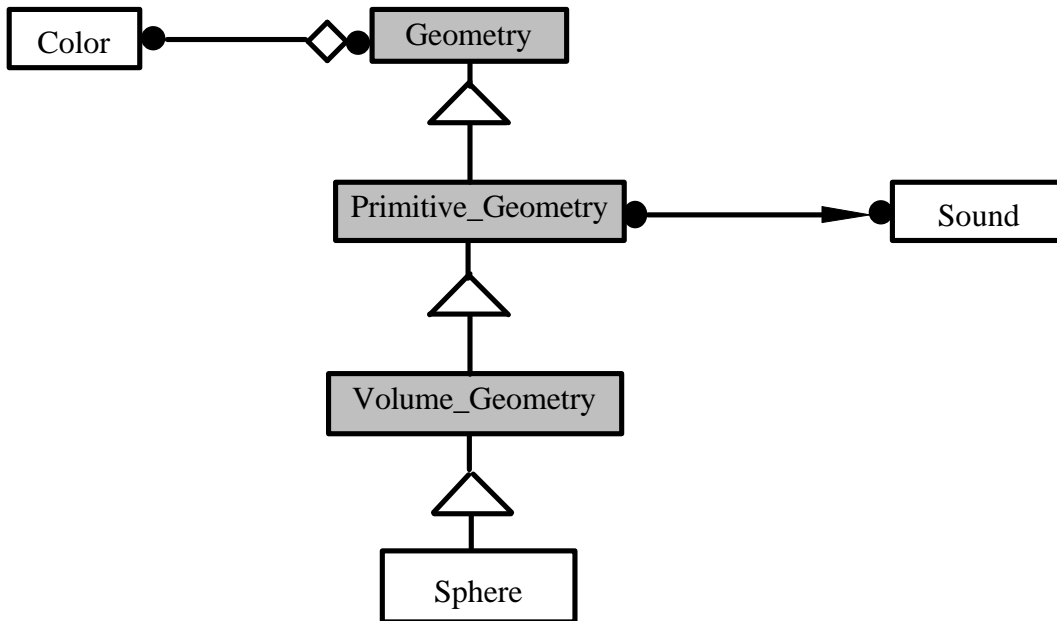


The aggregation relationship can also have multiplicity as described for the association relationship. The same symbology of open and filled circles is used with aggregation relationships:



This diagram shows that a Polygon Class is an aggregation of three or more ordered Vertex Classes. A Vertex Class is a component of zero or more Polygon Classes.

The following example shows how a piece of the SEDRIS Data Model would be interpreted:



This notation shows that a Sphere Class is-a Volume_Geometry; is-a Primitive_Geometry; and, is-a Geometry. So, the Sphere Class has inherited attributes from all its Parent Classes as well as has its own unique attributes. In addition, a Sphere Class has zero or more Sound Classes associated with it. The Sphere Class is composed of zero or more Color Classes. The Sphere Class “knows” that it may have a Sound Class, but the Sound Class does not know it is associated with a Sphere Class.

4. The SEDRIS Data Modeling Technique

As indicated earlier, data classes depicted in an object-oriented data model are an abstraction of real-world “things” found in the problem domain. By being able to analyze and design a software system using data with real-world terminology, improved understanding and communications can be realized. This is the tremendous benefit that the object-oriented methodology brings to software engineering. Using this modeling concept, you may expect to see in the SEDRIS Data Model such data classes as “tree”, “tank”, “building”, “berm”, etc. However, this is not the case. The problem domain of SEDRIS is the *representation* of the real-world environment. This representation is accomplished through the use of databases containing the data necessary to produce both visual and non-visual representations of synthetic environments. Therefore, one set of data classes found in the SEDRIS Data Model are the geometric data types used to depict a visual scene. These classes are data types such as polygons, lines, vertices, colors, textures, etc. The SEDRIS Data Model also contains a set of data classes that provide an alternative representation of the information contained in the geometric data types. These classes are the features data classes that use primitive data abstractions of node, edge, and face to describe point, linear, and areal data classes.

For example, one of the geometric data classes is the data class of a polygon. The polygon data class has attributes that include at least three vertices – which are also data classes. This relationship (a “has-a” relationship) is a rule that defines the polygon data type. If it doesn’t have at least three vertices, it is not a polygon. Vertex data types have additional attributes such as location and color. Taken together, this organization of data types and their attributes is a very clear definition of what constitutes a polygon and how to represent it in a visual scene. Therefore, unambiguous communication about a polygon has been made by proper use and interpretation of the model’s notation.

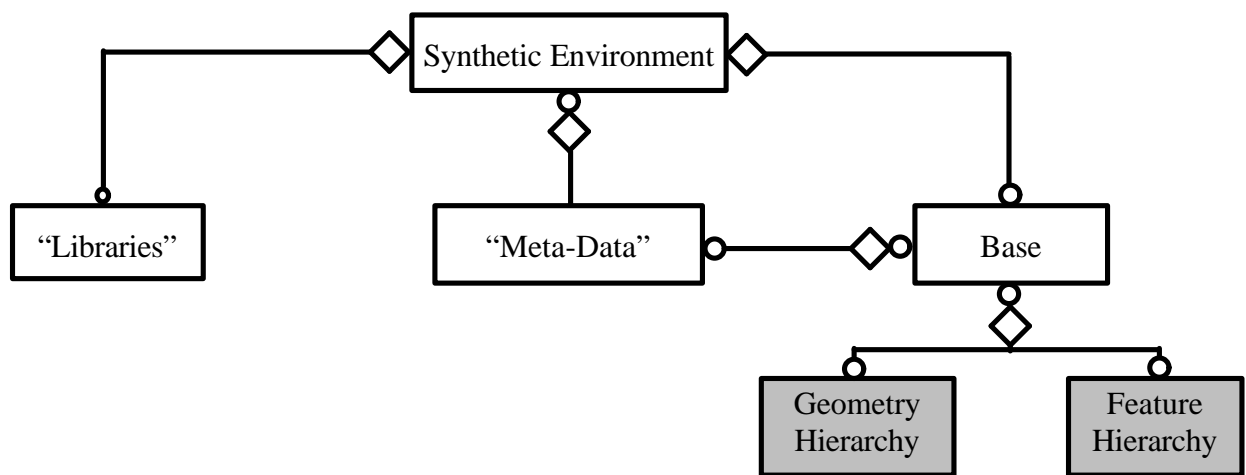
The polygon data class can contain the data used to create the visual representation of a “tree” or any other object in the synthetic environment. However, the data used to define a polygon that represents the image of a “tree” in a visual scene may not be the kind of data needed by a computer generated force (CGF) model to “see” the “tree” and make decisions based on the presence of a “tree” in the environment. To support the non-visual parts of simulation applications, the SEDRIS Data Model provides alternative data types that provide other representations of “things” in the environment. These alternative representations are implemented through the use of feature data classes. At the “tree’s” location in the environment, there is a point feature data class whose instance is identified as a “tree”. This point feature has certain attributes, described by its use of the primitive data classes of nodes, edges, and faces, that have been organized in a specific manner to define the representation of a “tree” feature model. This information can be directly used by the CGF models to make decisions about the “tree”. Should the CGF entity go around the “tree” or is the “tree” small enough to maneuver right over it? Can the CGF entity hide behind the “tree” or can the entity climb up in the “tree”? Note that the geometric description of the “tree” in polygons would not have been as readily used by the CGF models to determine the answers to the above questions. The alternative representation relationship between the geometric data to create the visual image of a “tree” and the feature data to reason about a “tree” is captured in the SEDRIS Data Model. This relationship is used to clearly define to a user the kind of “thing” to represent in the environment.

To further assist the reasoning models of the non-visual components of simulation applications, the SEDRIS Data Model also contains topological information about the “things” represented in a particular synthetic environment. The topology information is provided for both geometric and feature data types. The topology information provides connectivity and adjacency information about the spatial “things” in the environment. The topological information is contained in other specific data classes within the SEDRIS Data Model with relationships that tie this information to the geometric and feature data classes. The topological relationships assist the reasoning models by providing explicit information instead of performing calculations to derive the information.

5. How is the SEDRIS Data Model Organized ?

The SEDRIS Data Model is contained within a 16 sheet drawing. Although the sheets help in organizing the graphical presentation of the Data Model, they are not critical to understanding the model. What is critical is the organization and definition of the data contained in the model. This section is intended to present only a high level overview of the basic structure of the data model. The third document of the SEDRIS Documentation Set, *The SEDRIS User's Guide*, provides a detailed examination of the entire data model as well as guidance on its use.

The top level of the SEDRIS Data Model structure is summarized by the following illustration. All SEDRIS data transmittals contain an instance of the Synthetic Environment Class:

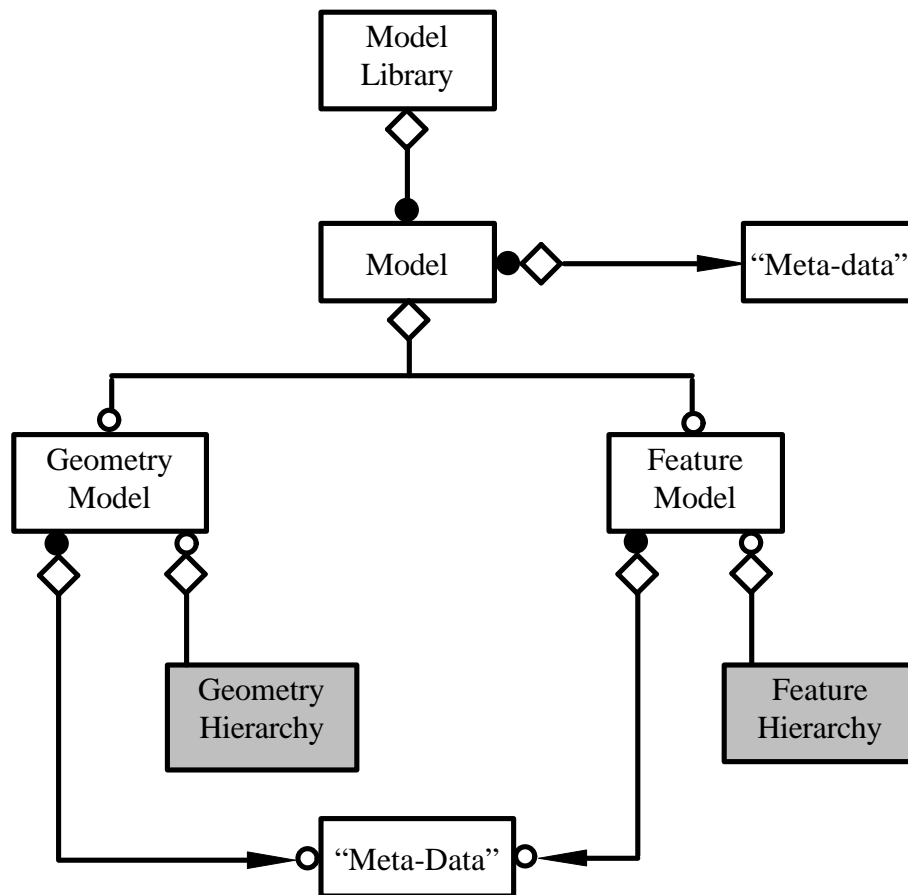


As can be seen from this diagram, the Synthetic Environment Class is an aggregation of other classes of data. The classes shown with their names in quotes are not real classes from the SEDRIS Data Model, but represent a category of classes in the Data Model and are shown here for illustration purposes only. The "Libraries" Class shown is really a set of library data classes such as Models, Colors, Symbols and Data Tables. In the SEDRIS Data Model, the "Meta-Data" Class shown above is in fact a set of data classes that provide a description of the transmittal, the points of contacts at the Producer's facility, keywords, and other identifying information. Again, "Libraries" and "Meta-Data" are not real classes used in the SEDRIS Data Model.

The Base Class, which has its own descriptive set of "Meta-Data" information, and is a true class from the Data Model, is the class defining the simulated world and its static features for the particular synthetic environmental database. The Base is an aggregation of Geometry Hierarchies and Feature Hierarchies (which are true classes in the Data Model). These Hierarchy Classes are abstract classes and each has an extensive set of subclasses. Their subclasses define all of the remaining data types, with their attributes, that define unambiguously the makeup of the transmitted synthetic environment. These subclasses also show the relationships between geometric data and feature data, indicating when one data type can be

used as an alternative representation of the other data type. In addition, topological relationships of both the geometric and feature data types are also contained within the data transmittal.

The following illustration shows another level of the Data Model. Again, like the first illustration, this diagram is not a true representation of all the classes and relationships that would be used to define this portion of the SEDRIS Data Model.



This diagram shows the Model_Library Class is an aggregation of zero or more Model Classes. A Model Class is composed of either a Geometry_Model or a Feature_Model. The Geometry_Model Class is an aggregation of the sub-classes of the Geometry_Hierarchy abstract class. Similarly, the Feature_Model Class is composed of sub-classes of the Feature_Hierarchy abstract class. The Model, Geometry_Model, and Feature_Model Classes each have a set of "Meta-Data" information describing them. Obviously, the Geometry_Hierarchy and Feature_Hierarchy Classes, since they are both abstract classes, have further levels of sub-classes that would be shown on other sheets of the Data Model.