SEDRIS Data Representation Model (DRM) Capabilities

https://www.sedris.org/drm.htm

ISO/IEC Technical Presentation Session

26 August 2019

Farid Mamaghani (farid@sedris.org)

About This Presentation

GOAL

To provide a quick overview of the capabilities available in the DRM

WHERE TO FIND MORE INFORMATION

The DRM is specified in ISO/IEC 18023-1.

Information on details of the DRM can be found at: *https://www.sedris.org/drm.htm*

Detailed tutorials on the fundamentals of DRM, as well as specialized uses of DRM for terrain, ocean, atmosphere and other representations are available at: *https://www.sedris.org/tutorials.htm*

WHAT TO EXPECT

Highlights of some of the key principles and concepts in DRM, which touches the relation to EDCS and SRM, understand. Some specific sub-topics related to data organization, 3D geometry data, organizing and representing tabular / grid data, and general data organization classes.

Assumptions and Conventions

- Familiarity with common environmental data and some of the techniques for modeling the environment.

- Familiarity with the SEDRIS technology components and how they fit together.

- Familiar with object-oriented design techniques and **UML notation**, such as **has-a**, **is-a**, **relations**, and **cardinality**.

- Brackets < > surrounding a phrase convey a class name
  Italic text inside brackets, or gray class box, indicates an abstract class (a class not meant to be instanced, but serves as a template for classes of its kind)

Outline

- Data Structures, Data Models, and Data Representation Model (DRM)
- Categories of DRM Classes/Functionality
- A Few Key Classes and Their Use in Data Representation
  - <Transmittal Root>
  - <Classification Data>, Attributes, EDCS
  - Basics of Representing Geometric Data and Primitives
    - <Polygon>, <Vertex>
  - Organizing Principles
  - Fundamentals of Tabular Representation
    - <Property Grid> related classes
- Summary
  - Where to find more information
- Refresher and Backup Material

Data Structures, Data Models

- A **data structure** is a specific arrangement of data elements and types, usually created for a specific application or purpose, that supports organizing, processing, retrieving, and storing the data.
- Generally, a **data model** is the specific collection of data structures and their relationships that specify a unique instance of data for some concept or object.
- However, **multiple data models** can be realized that describe the **same object or concept**.
- Problems in communication, interchange, and data processing arise because:
  – **Different data models** often reflect **different perspectives** on what is important about a concept
  – Software written to parse and process a specific data model's schema and format will not be able to process another data model

Example: Modeling a Tree

- Consider data model *A*, where a tree is represented using a data structure, as follows:
  - **height**
  - **species**
  - **stem diameter**
  - **location**

```
Tree_Rep
   Height
   Species
   Stem_Diameter
   Location
```

- Data model *B* may represent a tree using the following data structure:
  - **location**
  - **material properties of the trunk**
  - **stem radius**
  - **Height**

```
My_Tree
   Tree_Location
   Trunk_Material
   Stem_Radius
   Height
```

* For simplicity, assume same data type for same data element is used in both cases

Tree Example: Inherent Restrictions on A and B

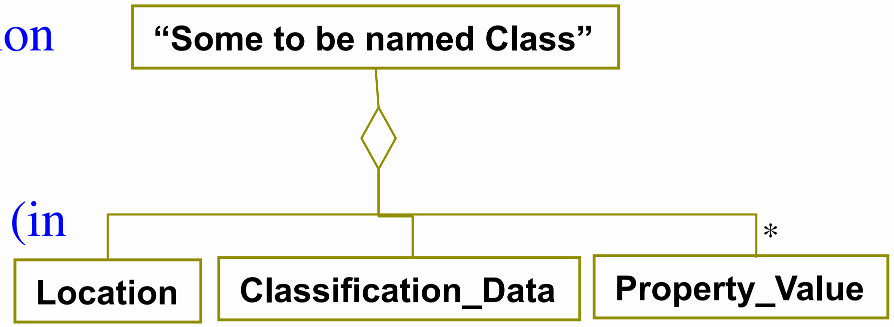
- Software designed for data model *A*'s data structure will not parse the values communicated by *B*.
- The semantic of "treeness" is built into the data structures of *A* and *B*.
- The semantic of what a field value means is specific to that data structure.
- If a decision is made to add more information about trees to either data model (e.g. foliage density), that data model's structures must be modified, along with the structure and logic of the software designed to process it.
- If a new kind of object (buildings, for example) is desired, another data model and data structures for that concept must be added, since "treeness" is inherent in this set of data structures.

Motivation for Designing a Data Representation Model

- **Not practical** to try to define data models for all possible types of data and their attributes.
- Need a different approach to design a model that can be applied easily to the many different kinds of environmental data, while still being general enough to be understood by many different applications.
- Key Principles
  - **Separating the semantics of what something represents from the "data primitives" used to represent it**
  - **Factoring out the common syntax and semantics of data models used to represent similar things**
- Combining these two principles gives us the tools to create a single schema to represent an endless variety of data models: the SEDRIS Data Representation Model.

Tree Example, Revisited

- The example described an object - a tree - that has a physical location and a collection of attributes, where the attributes differ depending on whether data model *A* or *B* is used to describe it.
- Consider some DRM class that can specify
    - a <Location> instance, specifying the location of the "thing" in the environment
    - a <Classification Data> instance that uses EDCS to specify what the object represents (in this example, a tree)
    - multiple and various <Property Value> instances, using EDCS, to specify properties of the object and the values of those properties



- Instances of such a DRM class can be used to define many kinds of objects.

What Is the SEDRIS DRM?

- A set of 306 classes, supporting a variety of different representation schemes, and the data types used to specify them

- The formal relationships between classes

- A set of constraints specifying requirements on instances of classes

- The classes fall into only a few broad categories when considered generally in terms of the functionality they supply

Broad Categories of Class Functionality

- **Organizers and Containers**

  Examples: <Transmittal Root>, <Environment Root>, <Library>, <Feature Hierarchy>, <Geometry Hierarchy>, <Time Related Features>

- **Primitives**

  Examples: <Polygon>, <Line>, <Point>, <Sound>, <Image>, <Areal Feature>, <Linear Feature>, <Point Feature>

- **Metadata**

  Examples: <Description>, <Keywords>, <Identification>

- **Modifiers**

  Examples: <Classification Data>, <Property Value>, <Control Link>, <Colour>, <Image Mapping Function>

How Is the SEDRIS DRM Expressed In Actual Data Products?

- Actual data sets contain *objects* - instances of DRM classes.
- The formal relationships between classes specify what relationships are allowed to exist between instances of those classes and what those relationships mean.
- The constraints further refine requirements for objects in specific contexts.
- A SEDRIS data or database is called a *transmittal*, and the objects contained within it are organized as a tree in terms of aggregate / component (whole vs. part) relationships between objects.

<Transmittal Root>

- Every transmittal (e.g., STF) contains exactly 1 instance of <Transmittal Root> as the 'root object' of the transmittal — the root of the object tree.

- Every object in a transmittal is connected by a path of aggregate/ component relationships with the <Transmittal Root> of that transmittal.

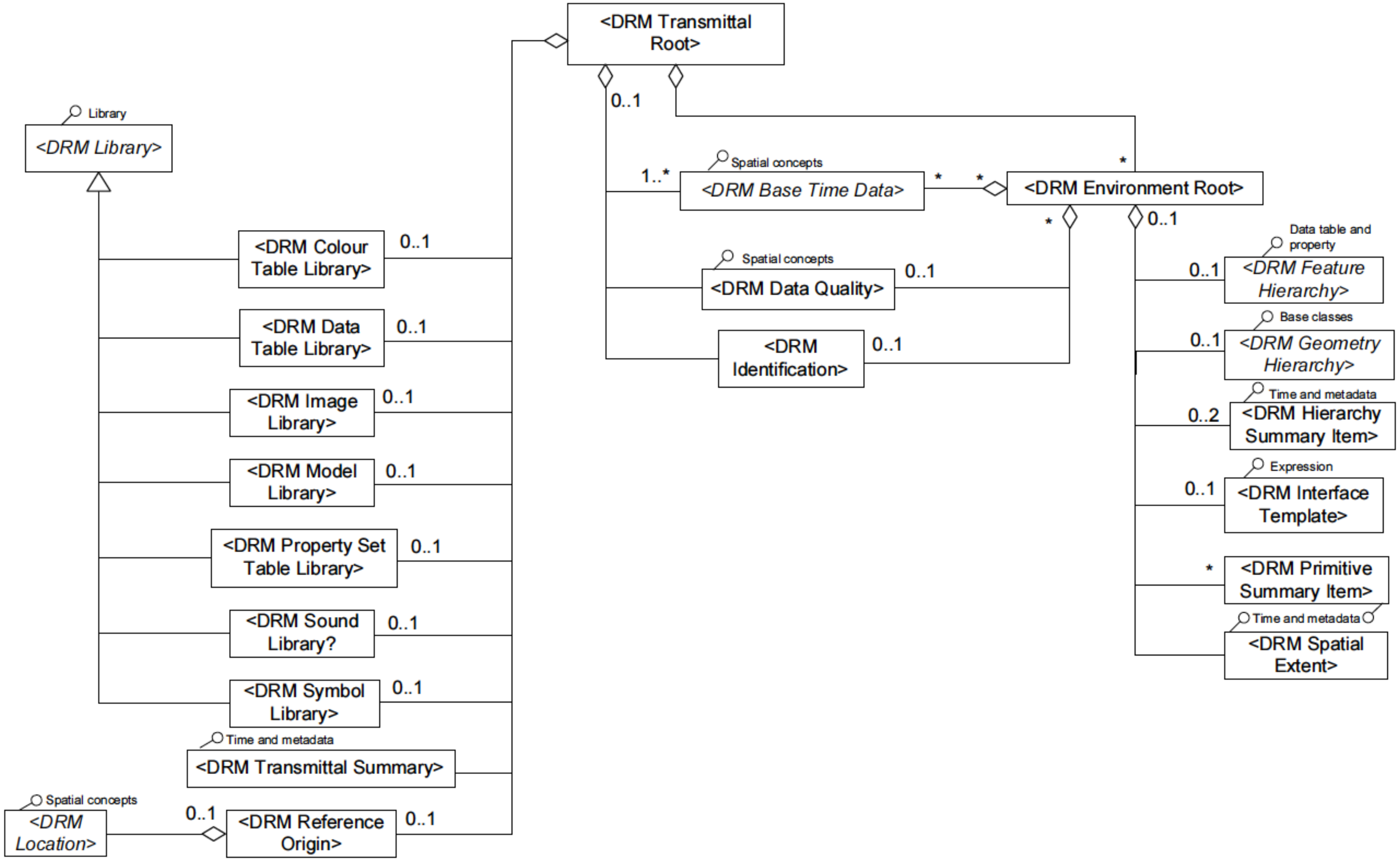
- Consequently, a <Transmittal Root> serves as the ultimate organizer of organizers for a transmittal.

What Does a <Transmittal Root> Organize?

- A <Transmittal Root> organizes:
  - **<Environment Root> instances** that **represent some environment**
  - **<Library> instances** that serve as repositories of representations of environmental **objects that tend to be shared and reused**
  - **metadata that applies to the entire transmittal**

- Any given <Transmittal Root> is required to supply some basic metadata, but otherwise data providers are free to use only those <Environment Root> and / or <Library> organizations that express their particular environmental data sets.

What Does a <Transmittal Root> Organize?



Sheet 1 — Transmittal structure (Version 4.1)

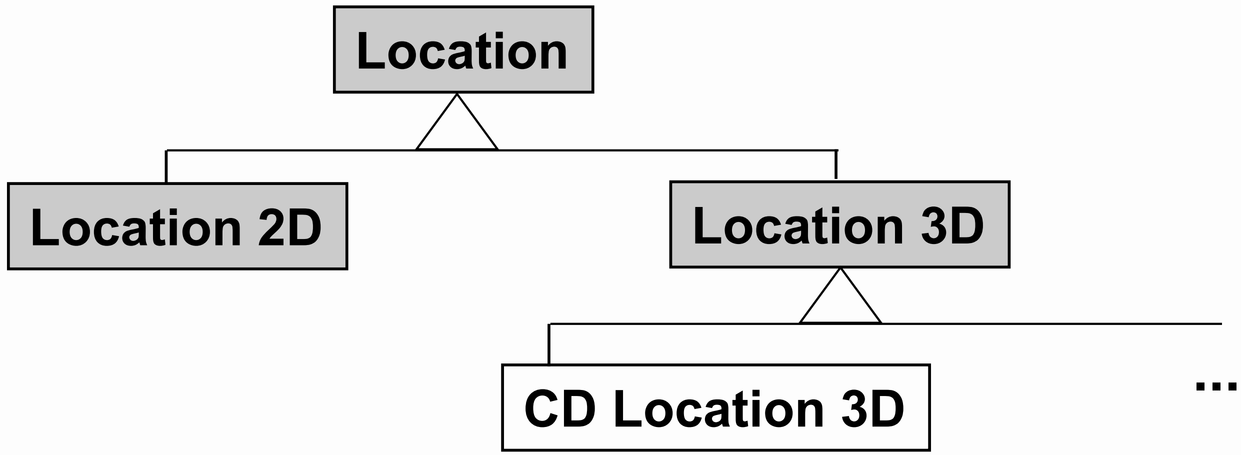
Fundamentals of Primitive Geometric Representation

What classes are needed for a simple geometric representation of an environment?

The classes covered in this section are

- **<Classification Data>**
- **<Environment Root>**
- **<Location>**
- **<CD Location 3D>**
- **<Polygon>**
- **<Property Value>**
- **<Spatial Extent>**
- **<Union Of Primitive Geometry>**
- **<Vertex>**

Spatial Reference Frames and <Location>



- All relationships in the DRM that need locations are specified with the <Location> class or its abstract subclasses. All the SRM's spatial reference frames are supported for all such relationships.
- The requirement that a given <Location> shall be valid in the spatial reference frame it appears in is specified as a constraint.
- In this example, the spatial reference frame 3D celestiodetic is used, which is show as: <CD Location 3D>.

<Classification Data> and EDCS

- A <Classification Data> instance attached to some object X specifies what the object tree rooted at X represents in the world - it *classifies* X.

- To do this, the tag field of a <Classification Data> instance specifies an EDCS Classification Code.

- <Classification Data> gives us the ability to represent many, many different environmental concepts unambiguously using the same type of transmittal organization.

Using EDCS Classification Codes

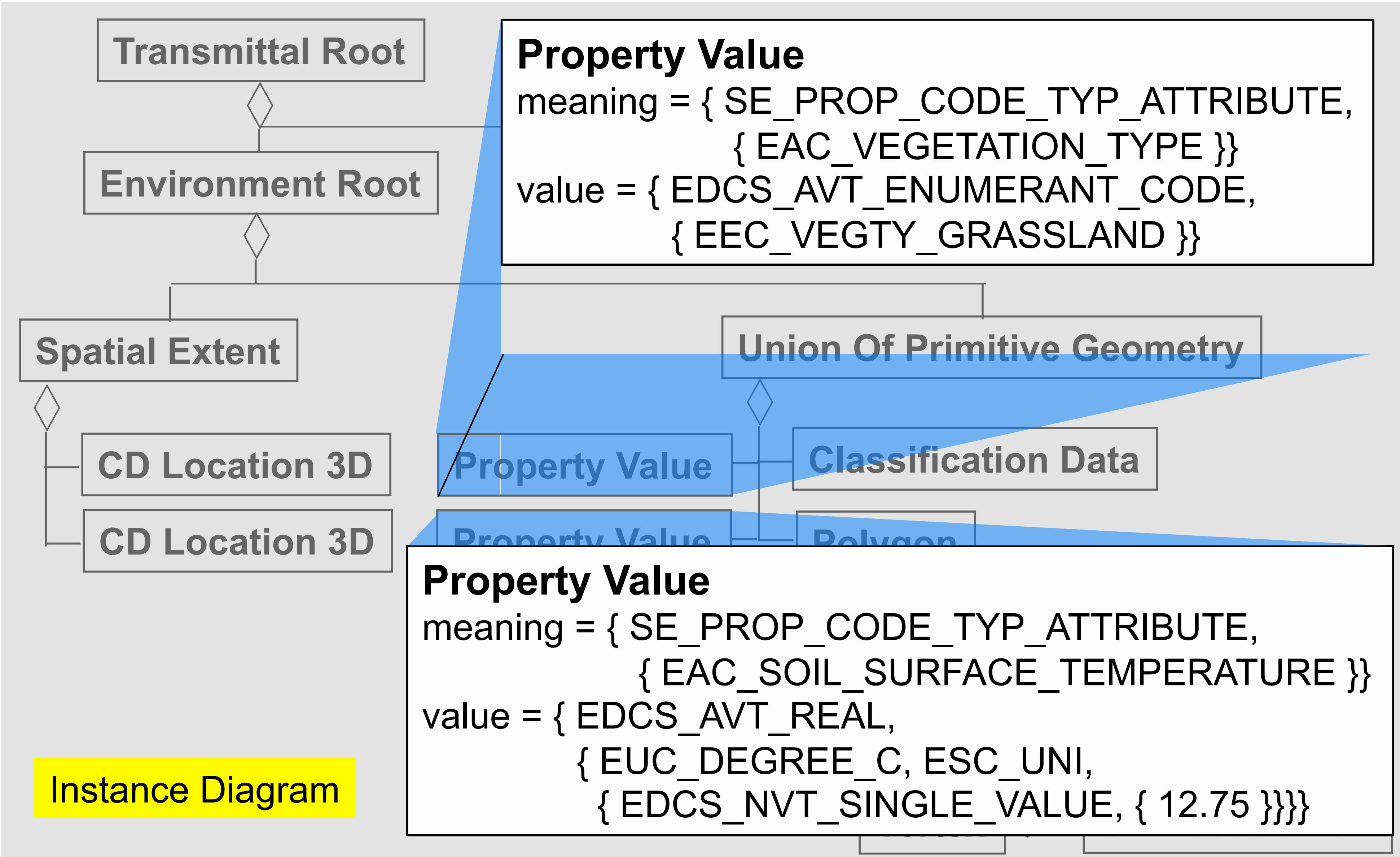
- The label (such as BENCH) is just that: a label for a concept.
- Don't confuse the label of an EDCS Classification Code with either
  - the integer code value stored in a transmittal, or
  - the definition of the concept being represented
- A transmittal actually stores the integer code corresponding to an ECC, not its label string. In software, use of the mnemonic constants provided with the EDCS implementation rather than raw integer codes is MOST STRONGLY recommended.
  - Greater clarity of software documentation
  - Code values may change between releases until EDCS moves up to the draft international standard level

Connecting the Representation with What It Represents

- A <Property Value> attached to some object X indicates the value of a *property* of the hierarchy of objects rooted at X - it describes an *attribute* of X.

- The meaning of a <Property Value> is specified with an SE_Property_Type - a "tagged union" data structure that may represent one of two kinds of codes:
  - EDCS_Attribute_Code (the most commonly used).
  - SE_Variable_Code, which is used primarily by the <Control Link> mechanism for specifying dynamic control, covered later.

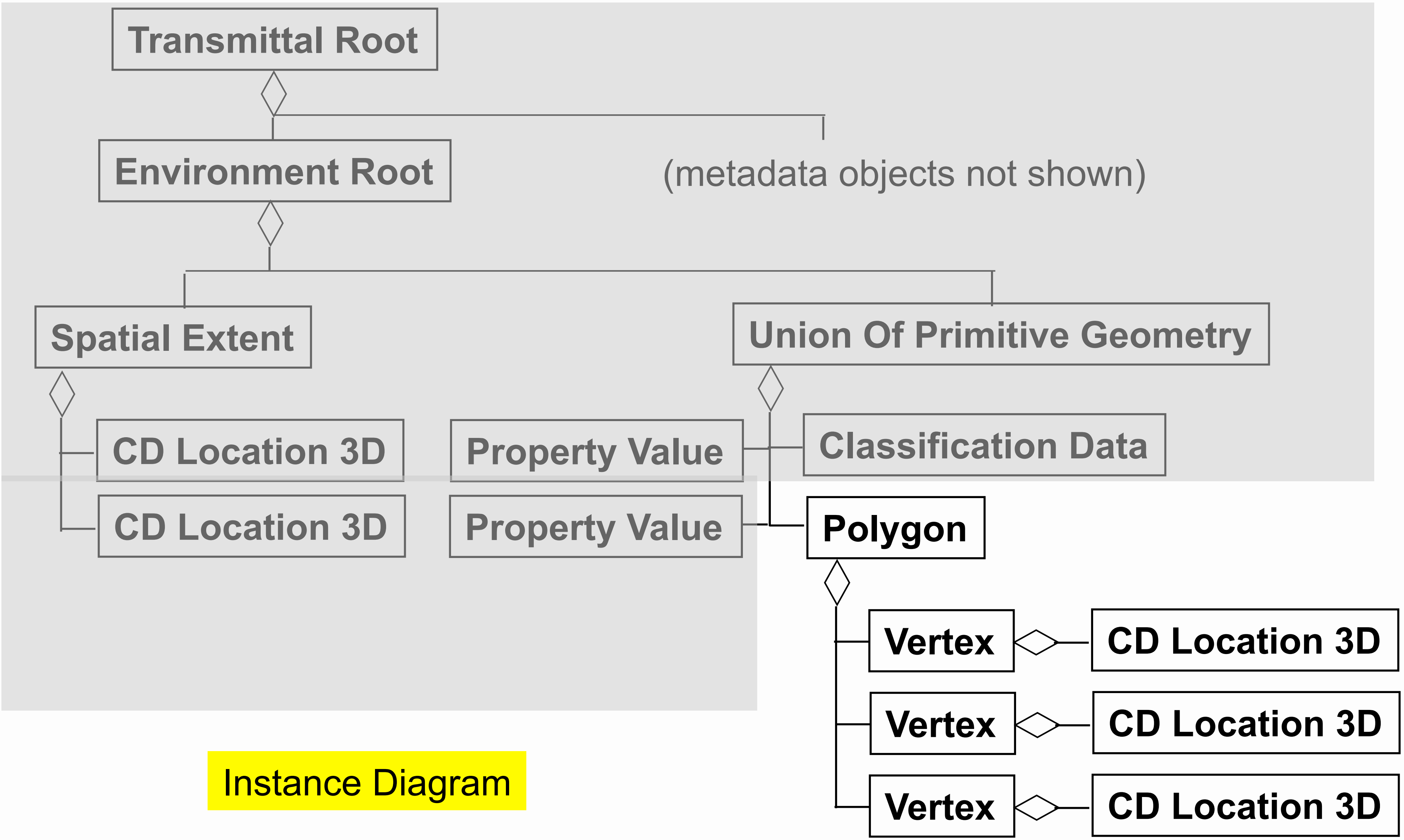
Some Environmental Properties



EDCS Attribute Codes

- An EDCS Attribute Code specifies some particular property of X, in contrast to an EDCS Classification Code, which indicates what X is in the environment.
  - ECC_TERRAIN specifies that X represents TERRAIN, while EAC_SOIL_SURFACE_TEMPERATURE specifies that a quantity represents the SOIL_SURFACE_TEMPERATURE for X.
  - ECC_WIND specifies that X represents a wind, while EAC_WIND_SPEED specifies that a quantity represents the wind speed for X.
- As with EDCS Classification Codes, don't confuse the label of an EDCS Attribute Code with its semantic meaning; check the definition.
- An EDCS Attribute not only has a definition, but is bound to an specific data type (real, integer, enumerated, Boolean, …); and some attributes require **units** for their meaning

The Primitives of the Representation: <Polygon>



Specifying a <Polygon>

- An instance of the <Polygon> class is required to have 3 or more vertices in counter-clockwise order, specifying a bounded portion of a plane.
- However, a <Polygon> may provide more information if desired.
- In this terrain example, suppose one of the <Polygon> components of the <Union Of Primitive Geometry> isn't just terrain, but a tract of with trees
- Can we classify that single <Polygon> instance as ECC_TREED_TRACT while leaving the rest of the union as plain ECC_TERRAIN?

Inheritance of 'Attribute' Components

- An individual <Polygon> can specify its own <Classification Data>, which can also be specified at the organization level by <Union Of Primitive Geometry>.
- A <Polygon> is said to *inherit* instances of certain classes from its aggregate, provided it does not 'override' those inherited components with components of its own. Some 'attribute' components that can be inherited are:

    <Classification Data> <Colour>

    <Property Value> <Image Mapping Function>

<Vertex>

- A <Vertex> is required to specify a <Location>, and is used as a component to create geometric primitives.
- However, like <Polygon>, <Vertex> is very flexible and can supply more than the minimum information if desired, such as <Colour> and
<Texture Coordinate> instances.
- As with <Polygon>, instances of the <Vertex> class can inherit 'attribute' components, provided they can be applied to a <Vertex>. For example, a <Vertex>
  - Does inherit <Colour>
  - Does not inherit <Classification Data>

Uniform Approach to Representation

- The principles shown for primitive geometric representation are the same basics principles for other representations in DRM.

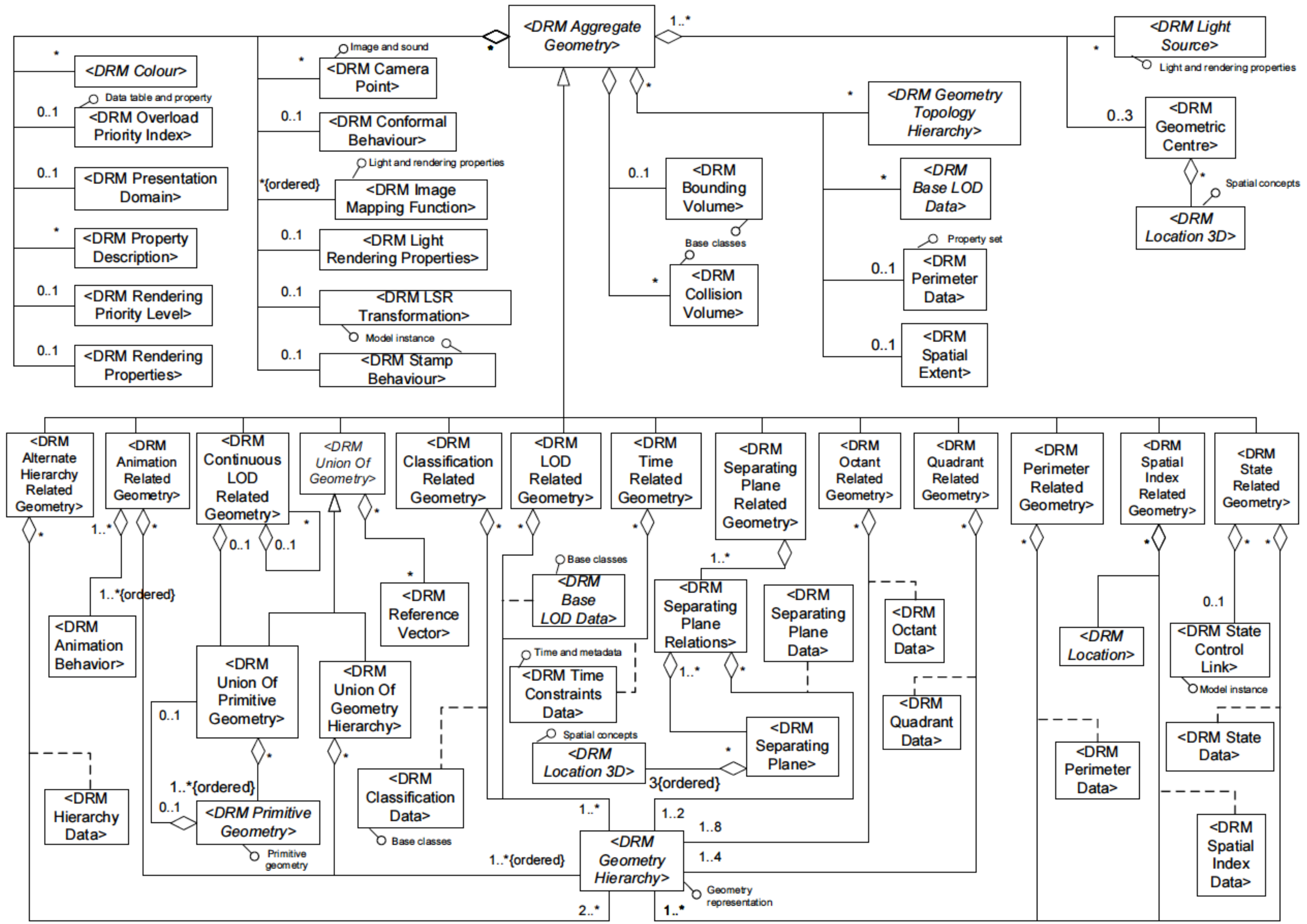
- The focus was on <Polygon>, organized under the <Union Of Primitive Geometry> class.

- But there is a total of 13 Geometry-specific organizing principles in the DRM that can organize instances of any of the subclasses of <Primitive Geometry>, including <Point>, <Line>, <Ellipse>, and <Finite Element Mesh>, in addition to <Polygon> that is the most commonly used.

- Same concepts apply to Features (GIS primitives) with 10 specific organizing principles. For details on organizing and using features refer to the ***Fundamentals of DRM*** tutorial.
- Both Features and Geometry are supported by their respective Topology (2D and 3D) primitives.

- Similar concepts apply to tabular (grid) data (e.g., weather, ocean volume, …) .

DRM Geometry Organizing Classes



Sheet 4 — Aggregate geometry (Version 4.1)

Representing Tabular / Grid Data

- Tabular Data used to capture properties and characteristics associated with an environmental object. <Property Table> classes are used for these. They are based on the <Data Table> class.
- Grid data is specifically bound to locations/position. Similar to <Property Table>, <Property Grid> related classes are used for these.
- What classes are needed for a tabular, or *gridded*, representation of an environment?

The classes introduced in this section are

<Property Grid Hook Point>

<Property Grid>

<Regular Axis>

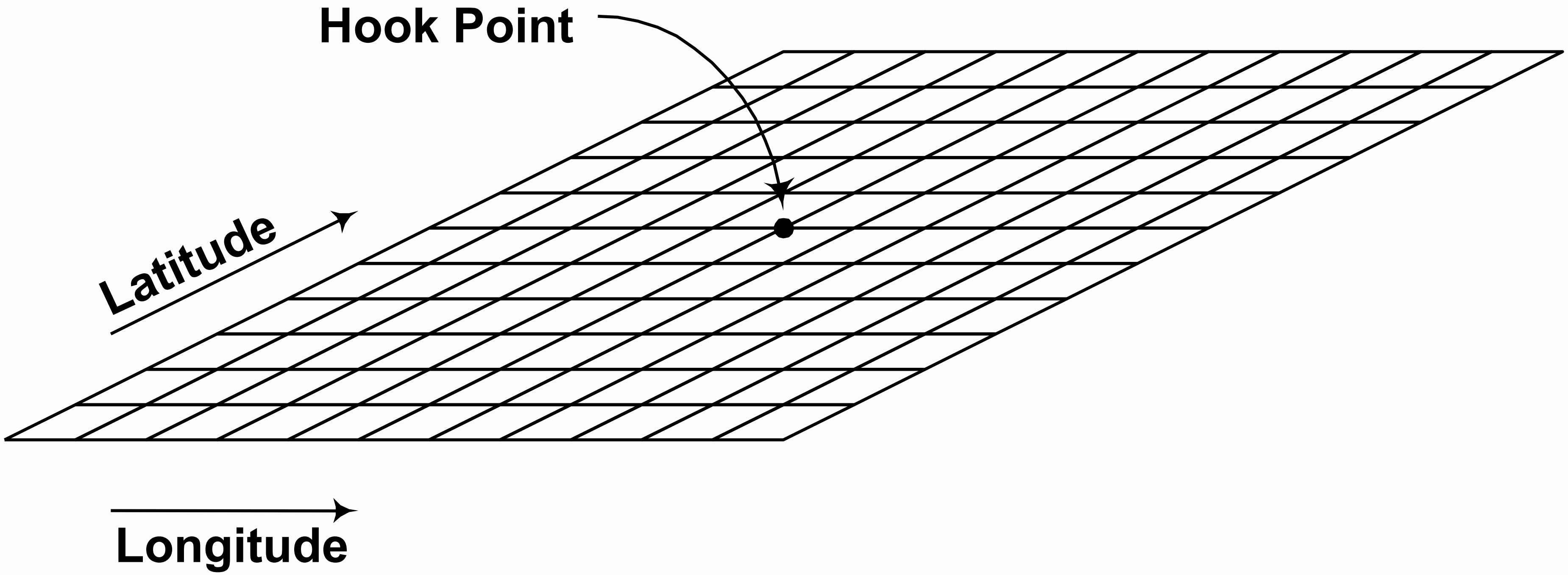
<Irregular Axis>

<Table Property Description>

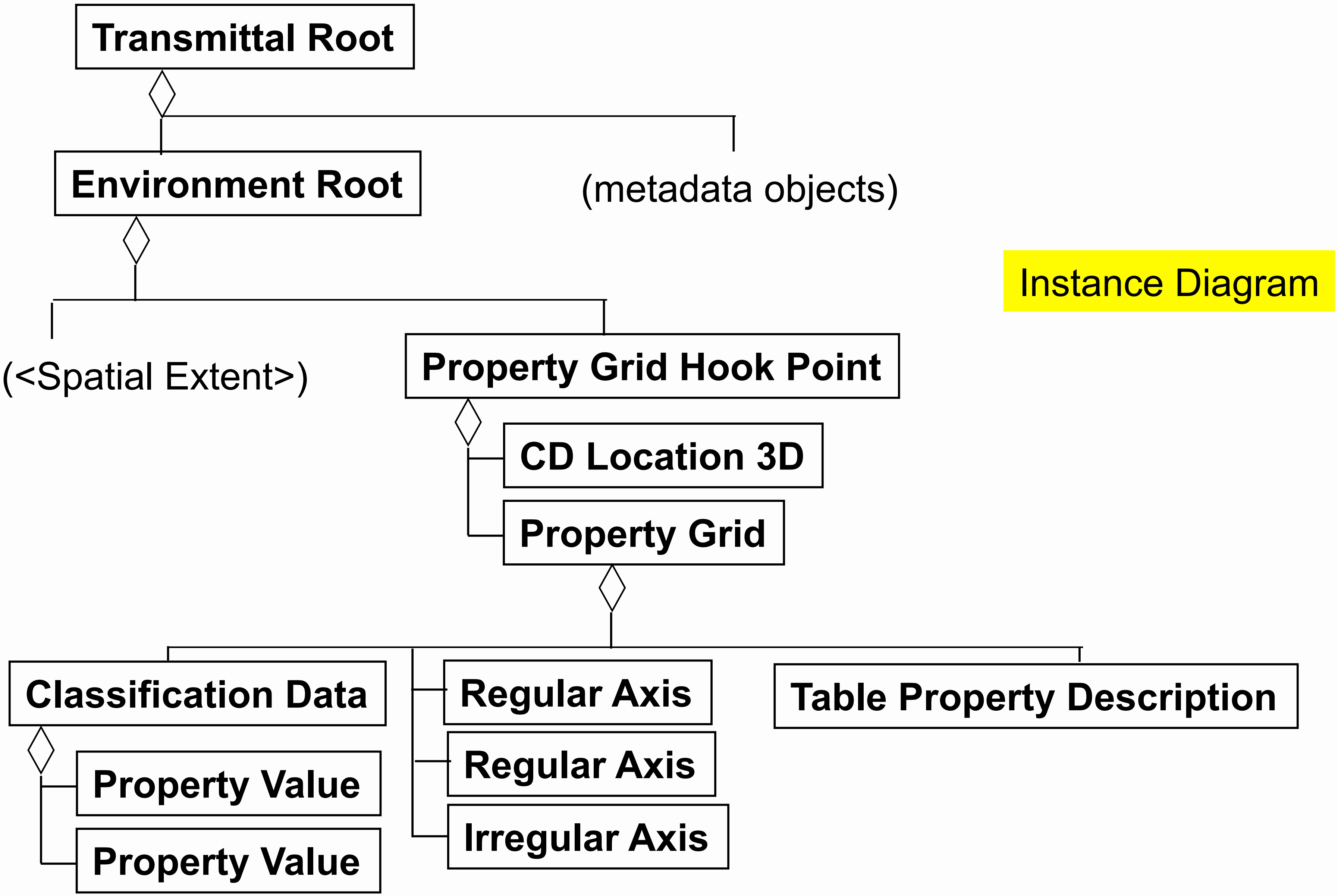
<Property>

Tabular Representation: <Property Grid>

A <Property Grid> instance specifies the values of one or more properties for each cell of a grid, which covers some region.



Tabular Representation: A <Property Grid> Example

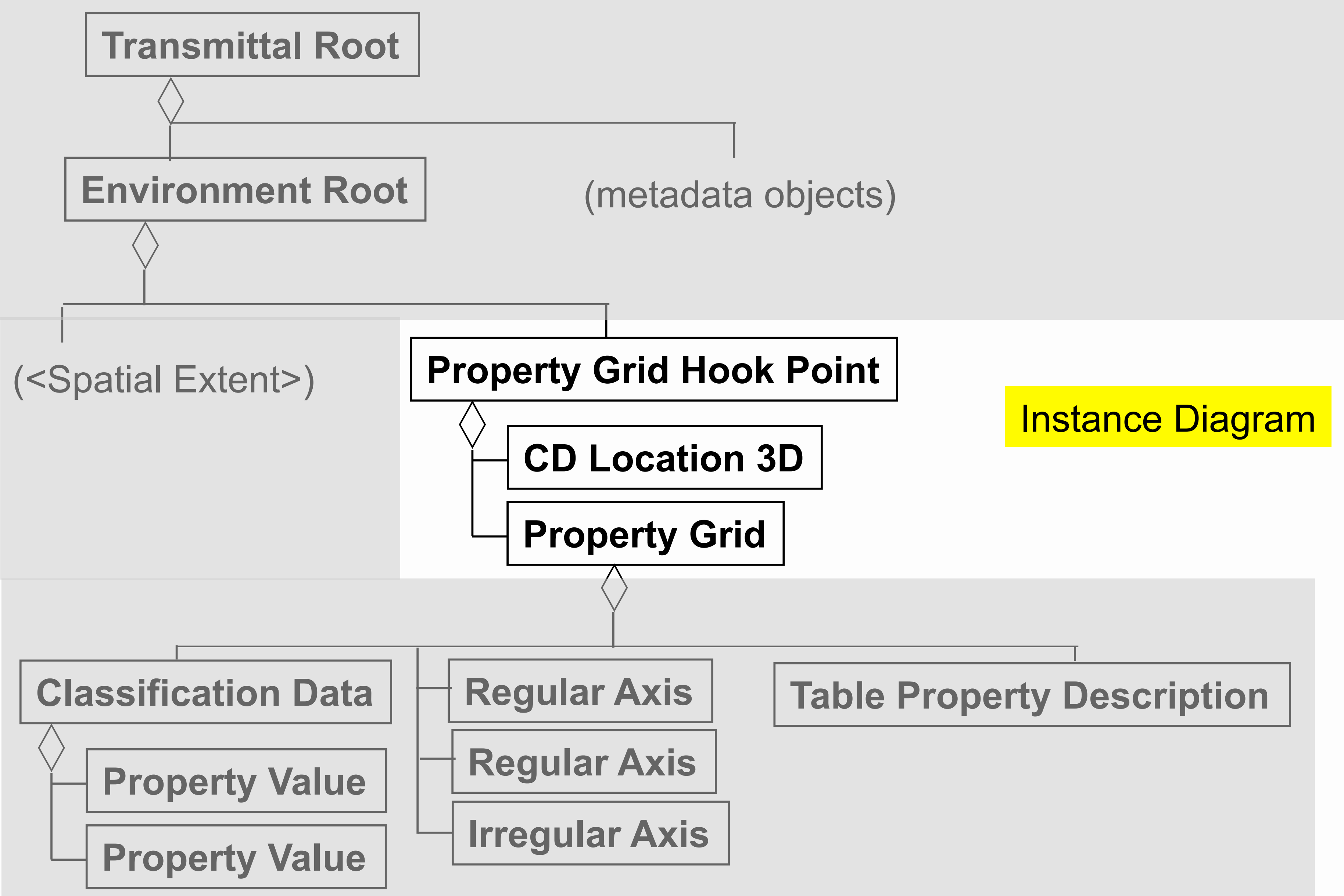


How <Environment Root> Represents the Environment

- **The geometric representation in this example is organized as a <Property Grid Hook Point>, which "hooks" its <Property Grid> components - each of which specifies its own spatial reference frame - to the spatial reference frame of the <Environment Root>.**

- **Remember that if all higher-level organization semantics are stripped away, all geometric organizations ultimately boil down to combinations of**
  - **<Union Of Primitive Geometry> (already covered)**
  - **<Property Grid Hook Point>**

<Environment Root>: The Environment Representation

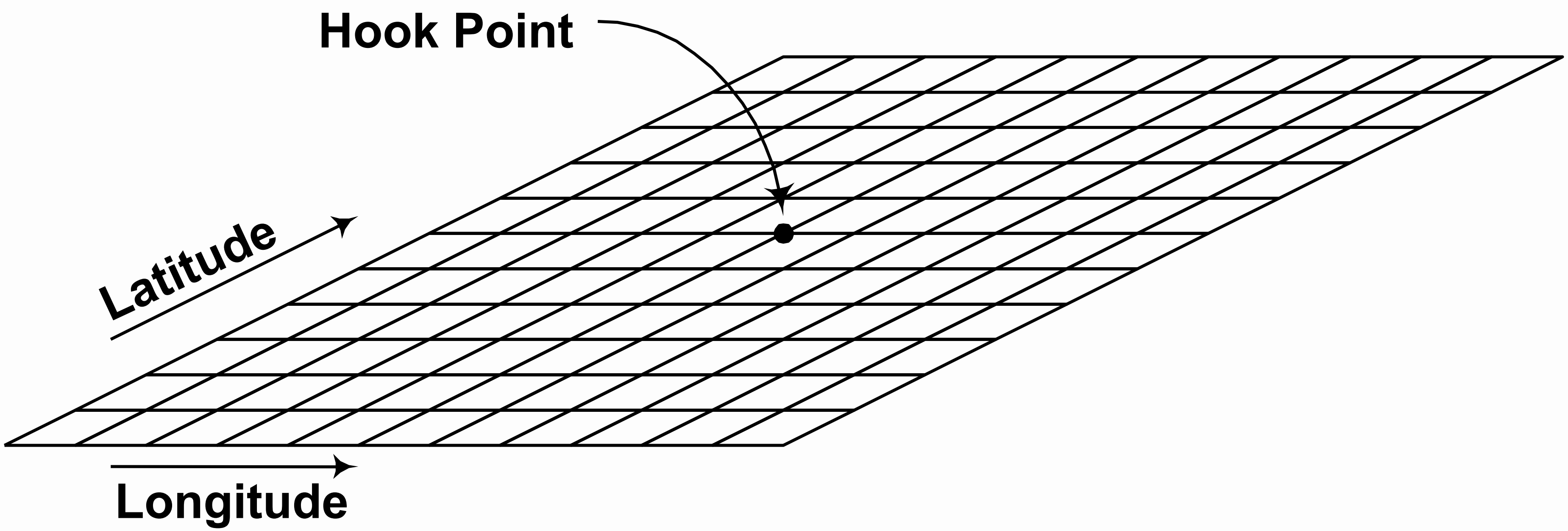


<Property Grid> and Spatial Reference Frame Issues

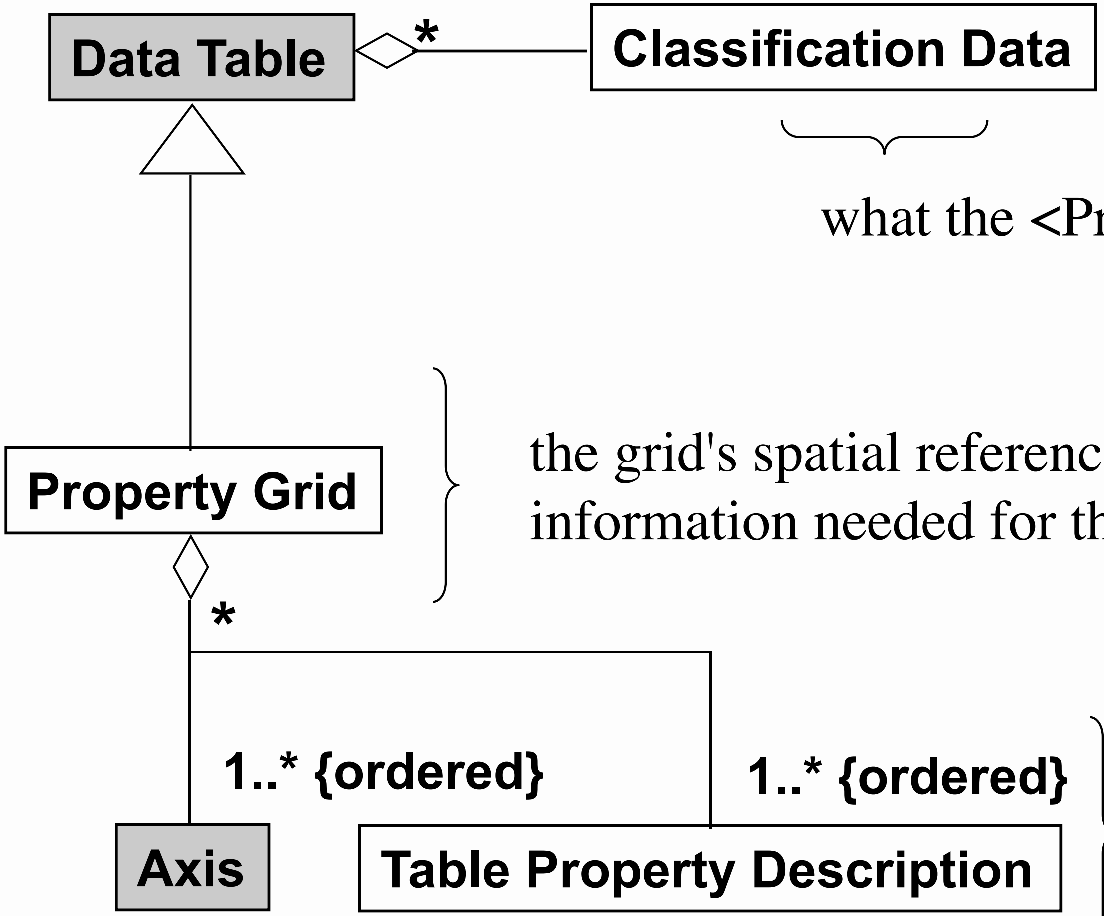
- **A <Property Grid> instance specifies its own SRF primarily to ensure that its "griddedness" is preserved if a consumer asks the API to convert the "native" SRF of part of a transmittal to another SRF.**
  - **During coordinate conversion/transformation between different spatial reference frames, <Axis> gridlines could become distorted so that they are no longer straight lines.**
  - **A <Property Grid> has its own SRF so that it stands apart from any coordinate conversion operations requested by the user.**
- **To connect its spatial reference frame to that of the <Property Grid Hook Point>, a <Property Grid> instance specifies two fields: spatial_axes_count and location_index.**

<Property Grid>: Setting the "Hook"

- The spatial_axes_count field specifies how many entries in location_index are significant.
- The location_index field indicates which point in grid space - some intersection of gridlines for the first (spatial_axes_count) <Axis> components - corresponds to the <Property Grid Hook Point>'s <Location>.



What Every <Property Grid> Instance Must Specify



what the <Property Grid> as a whole represents

the grid's spatial reference frame and the information needed for the "hook"

the properties for which values are being stored in the grid's cells

the structure of the grid cells, including the number and arrangement of cells

Detail, Class Diagram Sheet 6

<Regular Axis> and <Irregular Axis>

- **A <Regular Axis> specifies an <Axis> defined for a numeric attribute (specified by the axis_type), where the tick marks specified are regularly spaced, starting at some specified first value.**

- **An <Irregular Axis> is also defined for a numeric attribute, but the spacing of the tick marks is irregular, so the tick mark values are supplied by an array rather than specifying tick mark spacing.**

- **Detailed treatment of <Axis> instances, are provided in the *Advanced Application of the DRM* tutorials.**

More on Tabular Representation

- What can be represented?
  - Axis Types: Any EDCS attribute type
  - Cell Content: Any EDCS attribute type
- Arbitrary Dimensions (n-axes)
- Other capabilities
  - <Data Table> Reference Class
    - Allows entry into given row of table
    - Support for “physics based” modeling
  - Data Table in Library
    - Promotes re-use of tables
  - Supported by Metadata information

Further Topics on Tabular Representation

- <Data Table> has another subclass, <Property Table>, which provides much the same functionality as <Property Grid>, but for which none of the <Axis> components are spatial. While <Property Table> does not correspond to a representation of a spatial object, it can be used to represent other tabular information, such as material properties.
- The cell data of a <Data Table> instance is not part of the fields of the <Data Table>, but is an associated block of data accessed via the SEDRIS API, organized according to the layout specified by the <Axis> and <Table Property Description> components of that <Data Table>.

Higher-Level Organizing Principles

- **The higher level organizing principles for feature and geometric representation provide powerful mechanisms for expressing various kinds of semantic information.**

- **These organizing principles correspond to the subclasses of <Aggregate Feature>, for feature representation, and <Aggregate Geometry>, for geometric representation.**

Organizing Principles

- Union
- Alternate Hierarchy
- Spatial Index
- Quadrant
- Octant
- Perimeter
- Classification
- Level Of Detail
- Separating Plane (geometry only)
- Animation (geometry only)
- Time
- State

- Unless otherwise noted, all these organizing principles have at least 2 corresponding organizer classes, one for feature representations, the other for geometry.
- Three of them - union, spatial index, and perimeter - can also be applied to topological organizations.
- To understand a given principle for any one case - whether geometry, features, or topology - is to understand it for all cases.

Basic Organization Summary

- All geometric representation ultimately consists of organizations of <Union Of Primitive Geometry> and <Property Grid Hook Point> instances.
- All feature representation ultimately consists of organizations of <Union Of Features>.

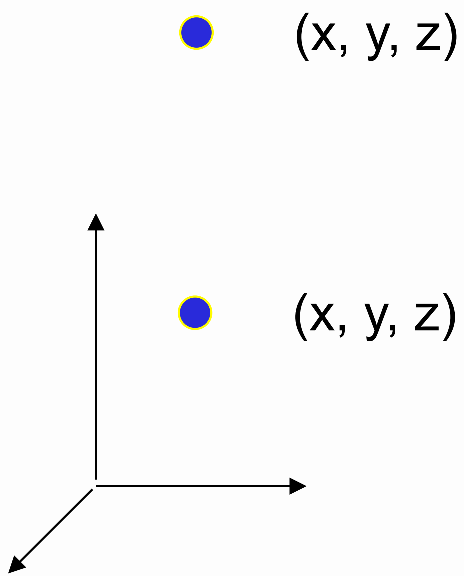
No matter how many and what kinds of extra levels of geometric or feature organization are present in a transmittal, they eventually boil down to organizing chunks of data in one of these representations.

Where To Go From Here

- **If interested in DRM details, review the DRM dictionary and HTML pages, along with the DRM class diagrams.**
- **Develop instance diagrams using the basic techniques discussed here and based on specific application topics of interest to you.**
- **Review the *"Advanced Application of the DRM"* tutorial, if you are interested in learning how to apply the DRM to solve more detailed environmental data problems (ocean, terrain, weather, and other data).**
- **Review the *"Advanced Use of the SEDRIS SDK"* tutorial, if you are interested in working with software to use, store, or manipulate DRM data.**
- **Review the *"How to Produce and Consume Transmittals"* tutorial, if you are interested in producing or consuming DRM data.**

Refresher and Backup Slides

What Is a Spatial Frame of Reference? (Cheat Sheet)

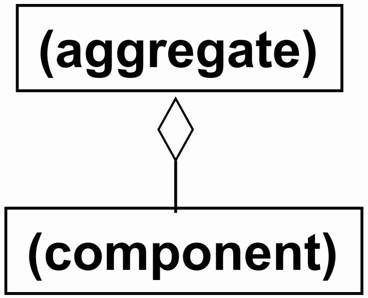


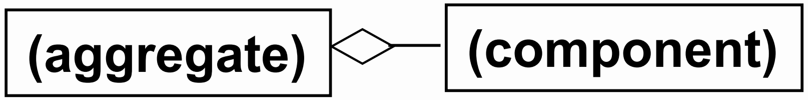
- *Coordinates* are pairs (2D) or triples (3D) of real numbers that designate the position of a point in a coordinate system.
- A *coordinate system* is a set of rules by which a coordinate can spatially relate a location to a unique coordinate system origin and associated axes.
- A *spatial reference frame* ties a coordinate system's origin to some Object Reference Model, such as a model of the Earth, so that it is no longer arbitrary but tied to the real world.

How the DRM Uses the SRM To Specify Spatial Data

- Coordinates are specified as <Location> instances or as coordinates within gridded data.
- Coordinates appear only in an object subtree rooted at an instance of some class that specifies a spatial reference frame, and must be defined in that spatial reference frame. For example,
  - Celestiodetic coordinates appear only in celestiodetic SRFs
  - 3D coordinates appear only in 3D SRFs
- <Environment Root> is one of the 5 classes of objects that specify spatial reference frames.
- Each such class has an srf_info field, defined using the SRM_SRF_Info structured type.

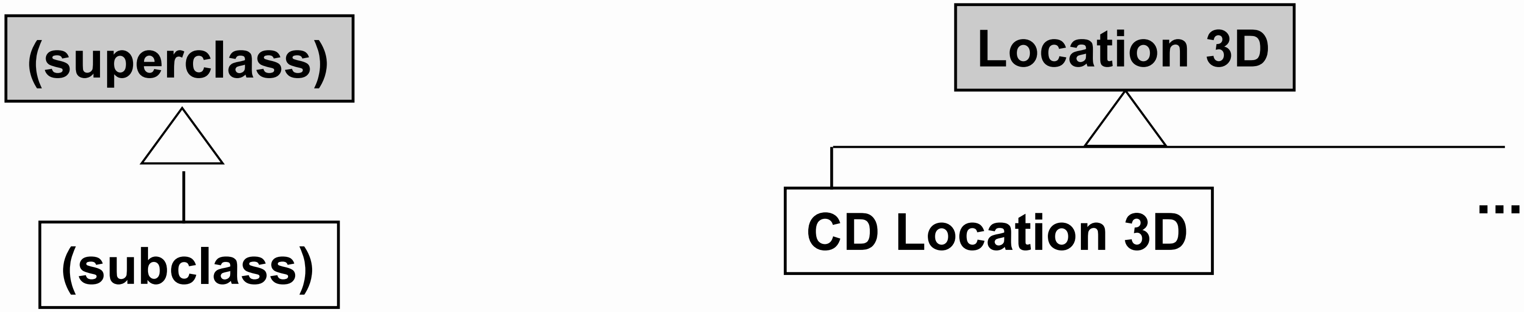
UML Notation Reminder: Has-A





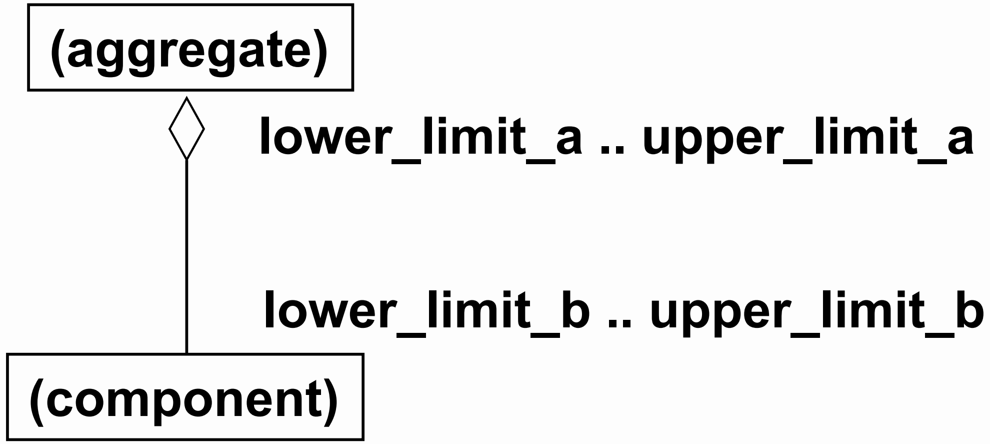
- **The has-a relationship relates two classes (or in a specific instance, two objects) in the roles of *aggregate* (the greater whole) and *component* (a part). The diamond, rather than drawing order, indicates which participant in a relationship is in the aggregate role.**
- **The DRM specifies which classes can participate in what relationships and playing what roles.**

UML Notation Reminder: Is-A



- An is-a relationship between 2 classes indicates that one is a subclass of the other - that any object belonging to the subclass is also a member of the superclass. The triangle, rather than drawing order, indicates which is the superclass.
- All superclasses in the DRM are *abstract* - they exist only to abstract common characteristics of their subclasses. They are shaded on class diagrams.
- Only *concrete* classes may have actual objects, so only concrete instances appear in instance diagrams.

UML Notation Reminder: Multiplicity and Ordering



- The *multiplicity* of a class relationship indicates, for an instance of either class, how many instances of the other class it may be related to.
  - A range is specified as [lower limit]..[upper limit], where an asterisk for the upper limit indicates that there is no upper limit.
  - An asterisk by itself means "zero or more".
  - For an exact number (lower_limit = upper_limit), the number is given.
- When more than one component may be present and the order of the component instances carries semantic information, the component end of the class relationship is marked with {ordered}.