

Fundamentals of the DRM

<http://www.sedris.org/drm.htm>

SEDRIStm Technology Conference
Lake Buena Vista, Florida
6 January 2004

Michele L. Worley
Science Applications International Corporation
michele.l.worley@saic.com

About This Tutorial

DESCRIPTION

The SEDRIS DRM provides a common representation model to allow users both to specify their own environmental data unambiguously, and to understand others' data clearly. The DRM specifies a set of classes, including the formal relationships between them and applicable constraints, thus ensuring that the syntax and structural semantics of the data are fully expressed and correctly understood by users. This tutorial provides a complete introduction to the SEDRIS DRM. It includes: a review of the notation used; the DRM organization and usage with sample applications; how the DRM utilizes the Environmental Data Coding Specification (EDCS) and the Spatial Reference Model (SRM); and a description of key concepts as represented by DRM classes, such as metadata, libraries, topology, point-sampled and grid data, organizational schemes, attributes, features, and geometry.

WHO SHOULD ATTEND

Environmental modelers interested in using SEDRIS, software engineers who plan to implement applications based on SEDRIS technologies, and those interested in gaining a better appreciation for the most fundamental SEDRIS technology, the DRM.

PREREQUISITE

Prior knowledge of object-oriented design and key issues in representation of environmental data is strongly recommended. Prior attendance at either the "Introduction to SEDRIS for Managers" or the "Fundamentally SEDRIS: The Technology Components" tutorial is recommended.

WHAT TO EXPECT

At completion, the attendee will be able to read and understand the DRM, the rules defined and imposed by the DRM, the use of the EDCS and SRM, and the use and organization of the data classes in the DRM.

Prerequisite

To get the most from this tutorial, we assume you know the following information as a prerequisite to this session:

- A solid understanding of the SEDRIS technology components and how they fit together. We assume you have attended *“Introduction to SEDRIS for Managers”* or *“SEDRIS: The Technology Components”*.
- Familiarity, or at least a basic knowledge, of common environmental modeling techniques, common environmental data, and data representation approaches.
- A basic understanding of object-oriented design techniques.

Tutorial Outline

- **What Is a Data Representation Model?**
 - **What Is the SEDRIS DRM?**
 - **How Is the SEDRIS DRM Expressed in Actual Data Sets?**
- **Key Classes and Their Use in Data Representation**
 - **<Transmittal Root>**
 - **<Environment Root>**
 - **Fundamentals of Geometric Representation**
 - **Fundamentals of Feature Representation**
 - **Higher-Level Organizing Principles**
 - **<Library>**
 - **Metadata**
- **Summary**
 - **Where to go from here**

What Is a Data Representation Model?

- **The SEDRIS data representation model is more than a simple abstraction of a given data model.**
- **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 and interchange arise because**
 - **different data models may 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 that used to realize another data model**

Example: Modelling a Tree

- **Consider data model A, where a tree is represented using a data structure containing**
 - height
 - species
 - stem diameter
 - location
- **Data model B may represent a tree using a data structure containing**
 - location
 - material properties of the trunk
 - stem radius
 - height

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 built into the 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

- It isn't 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 describes 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 the tree.
- 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 (a tree)
 - 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 classes and the data types used to specify them**
- **The formal relationships between classes**
- **A set of constraints specifying requirements on instances of classes that cannot be captured solely by the syntax of the relationships between classes**

The Classes of the DRM

The DRM contains 303 classes, supporting a variety of different representation schemes...

...but 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>

- **Primitives**

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

- **Metadata**

Examples: <Description>, <Keywords>

- **Modifiers**

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

How Is the SEDRIS DRM Expressed In Actual Data Sets?

- 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 set 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 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.**

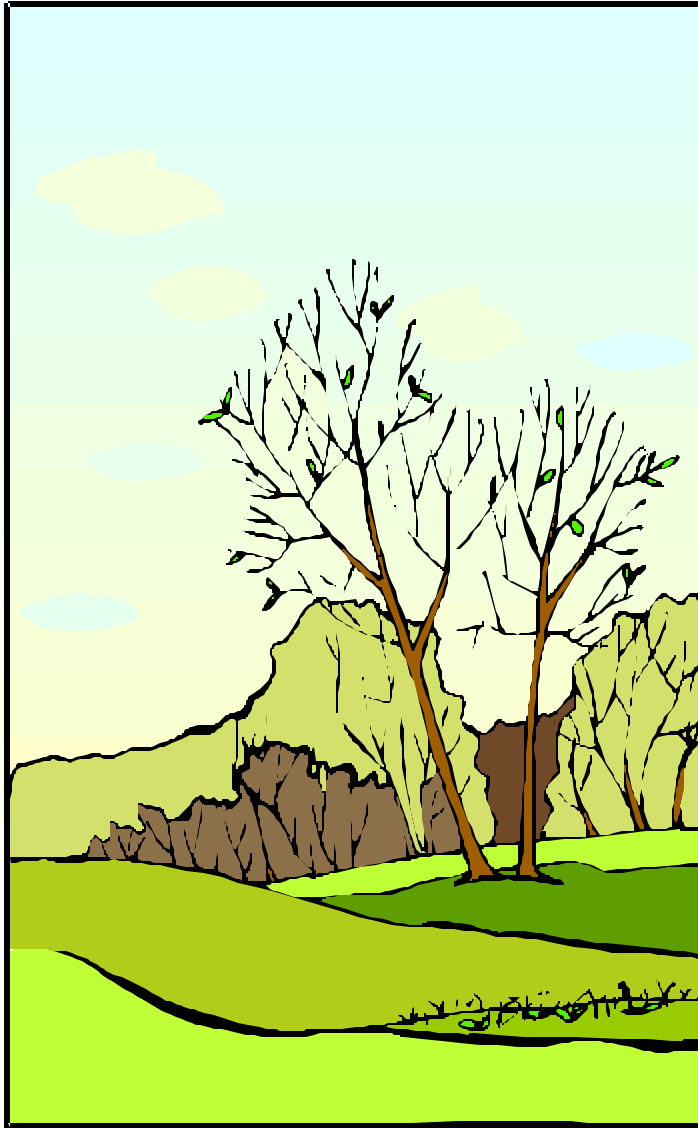
<Transmittal Root> Organization Topics

- **<Environment Root>**
 - We'll discuss the key classes needed for <Environment Root> to represent an environment, illustrated by examples.
 - After covering the lowest-level organizing semantics of such representations and the primitives they organize, we'll discuss the remaining organizing principles.
- **<Library>**
 - The functionality of some key classes related to the <Library> mechanism, with examples.
- **Metadata**
 - We'll discuss the key classes needed to support the metadata that <Transmittal Root> instances must provide, and how consumers can use some of that information.

<Environment Root> Topics

- **We begin by defining the term spatial reference frame, and showing how the DRM organizes spatial data.**
- **We apply this knowledge, showing the functionality provided by an <Environment Root> to support geometric and feature representations of an environment, as well as presenting the fundamentals of such representations, supported by examples.**
 - **Fundamentals of primitive geometric representation**
 - **Fundamentals of tabular representation**
 - **Fundamentals of feature representation**

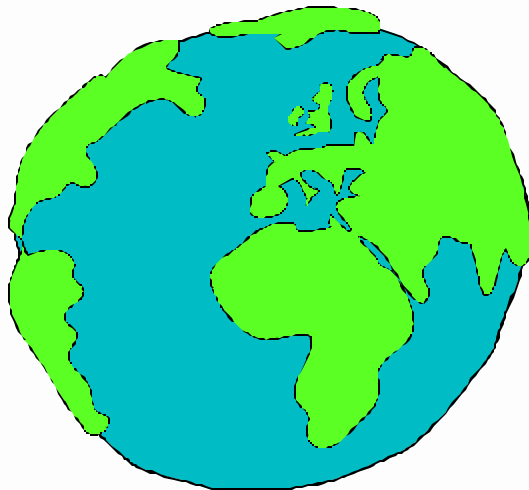
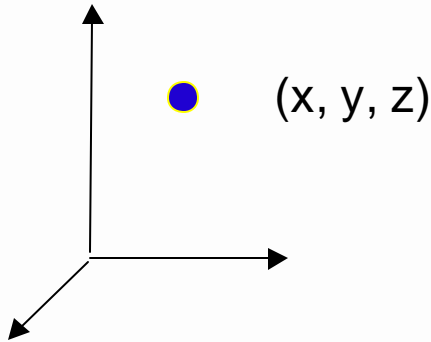
<Environment Root>



- An <Environment Root> instance specifies an environment - a 'world' or 'playbox' - in a given transmittal.
- Every <Environment Root> instance must specify
 - a spatial frame of reference
 - a bounding box specifying the spatial domain covered by the environment representation
 - the actual environment representation
- How is this information represented?

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

● (x, y, z)



- **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.**

<Environment Root> Topics

- We begin by defining the term spatial reference frame, and showing how the DRM organizes spatial data.
- We apply this knowledge, showing the functionality provided by an <Environment Root> to support geometric and feature representations of an environment, as well as presenting the fundamentals of such representations, supported by examples.
 - **Fundamentals of primitive geometric representation**
 - Fundamentals of tabular representation
 - Fundamentals of feature representation

Fundamentals of Primitive Geometric Representation

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

The classes covered in this section are

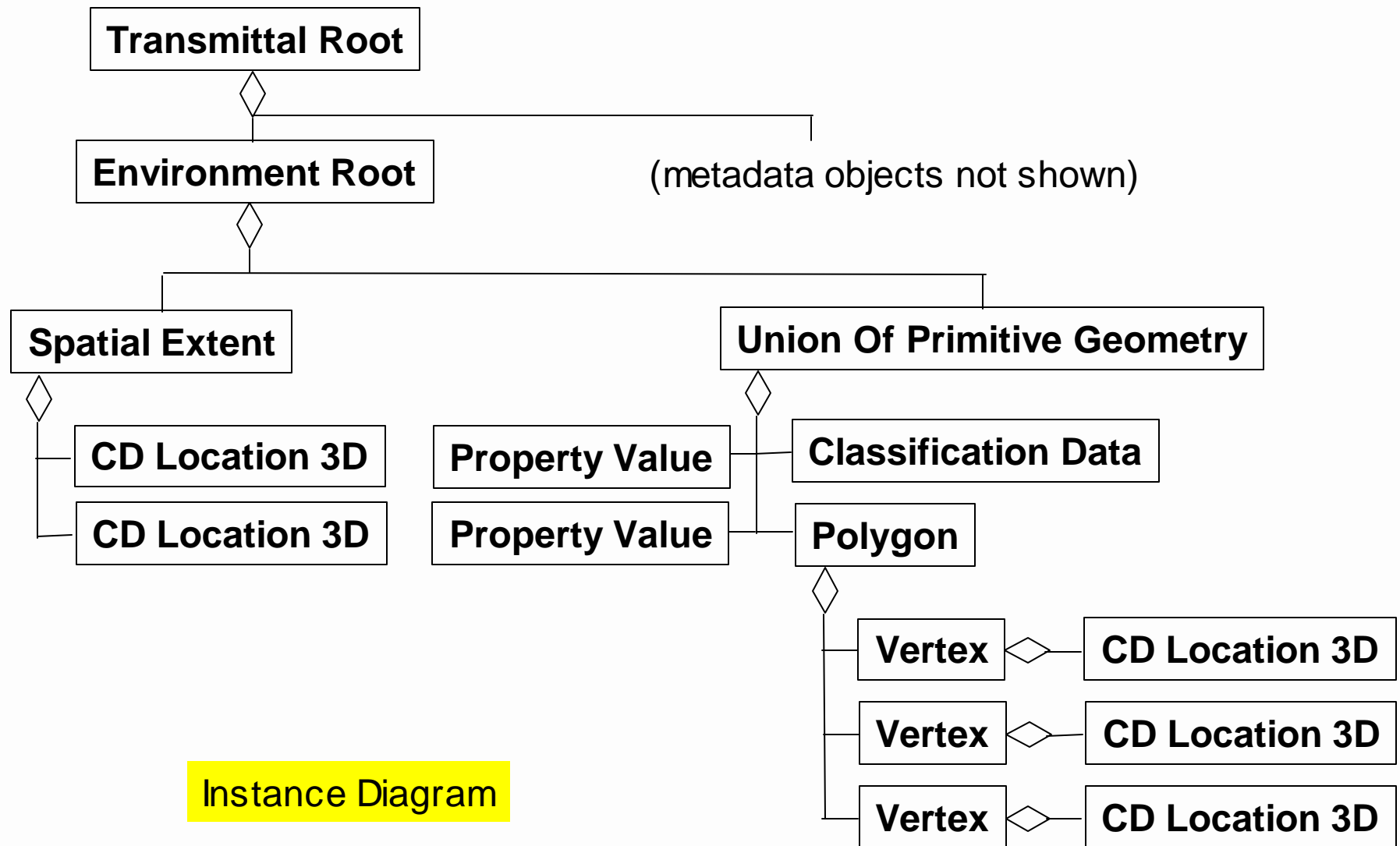
- <Classification Data>**
- <Environment Root>**
- <Location>**
- <CD Location 3D>**
- <Polygon>**
- <Property Value>**
- <Spatial Extent>**
- <Union Of Primitive Geometry>**
- <Vertex>**

UML Notation Reminder: Has-A

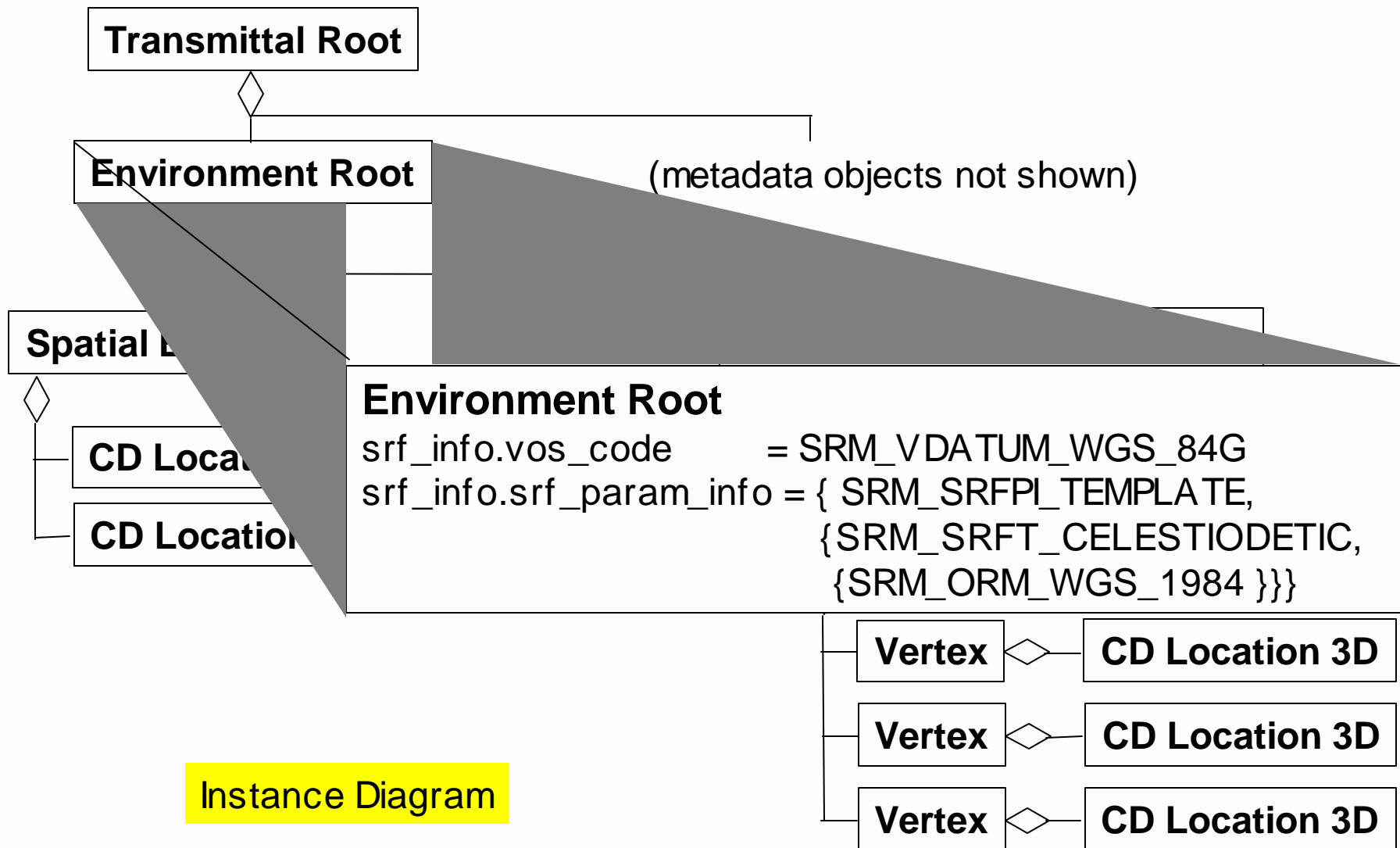


- The has-a relationship relates two classes (or in a specific instance, two objects) in the roles of **aggregate** (the greater whole) and **component** (a part). The diamond, rather than drawing order, indicates which participant in a relationship is in the aggregate role.
- The DRM specifies which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

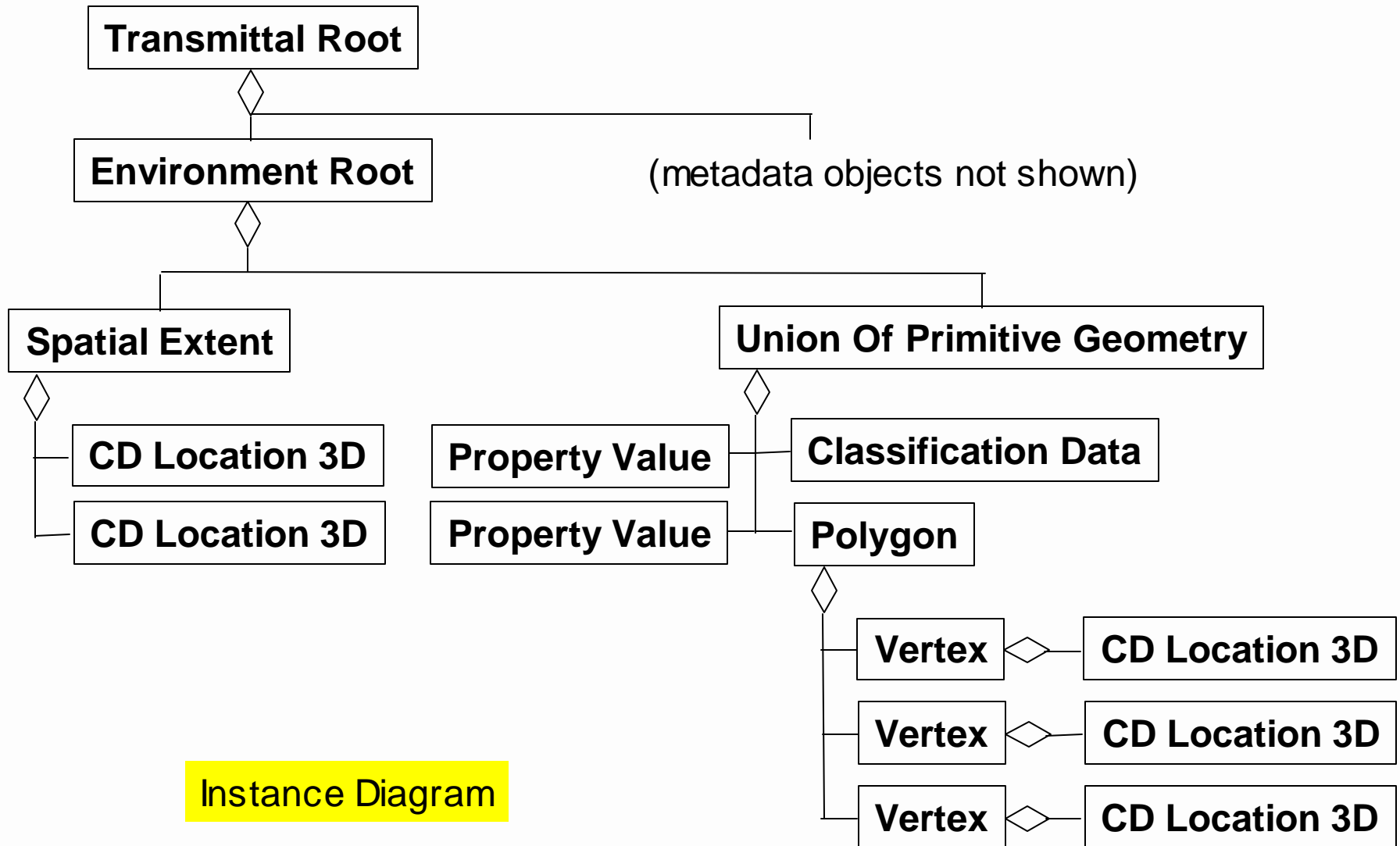
Organizing Primitive Geometry: An Example



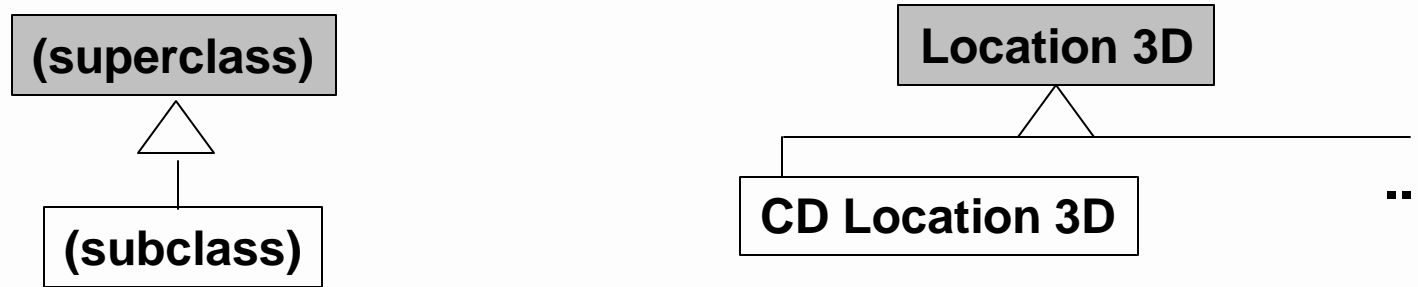
<Environment Root>: Specifying SRF Parameters



Specifying Individual Locations

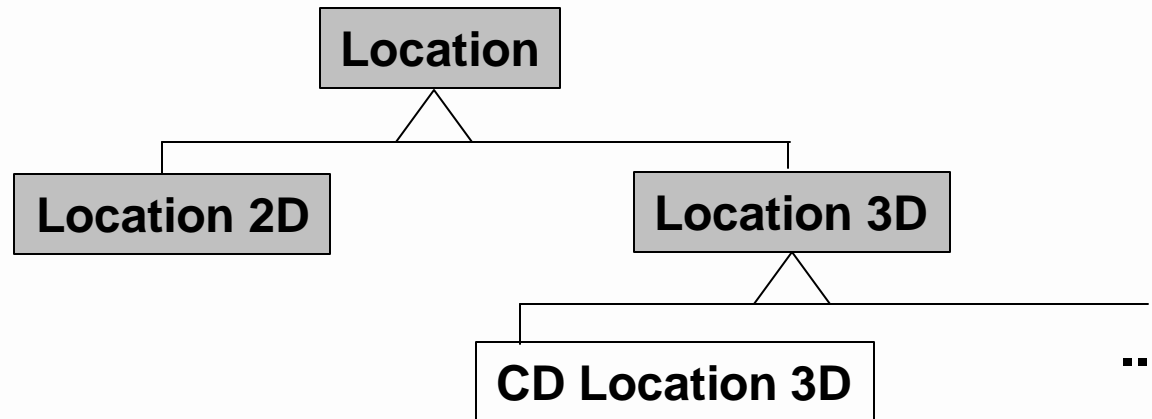


UML Notation Reminder: Is-A



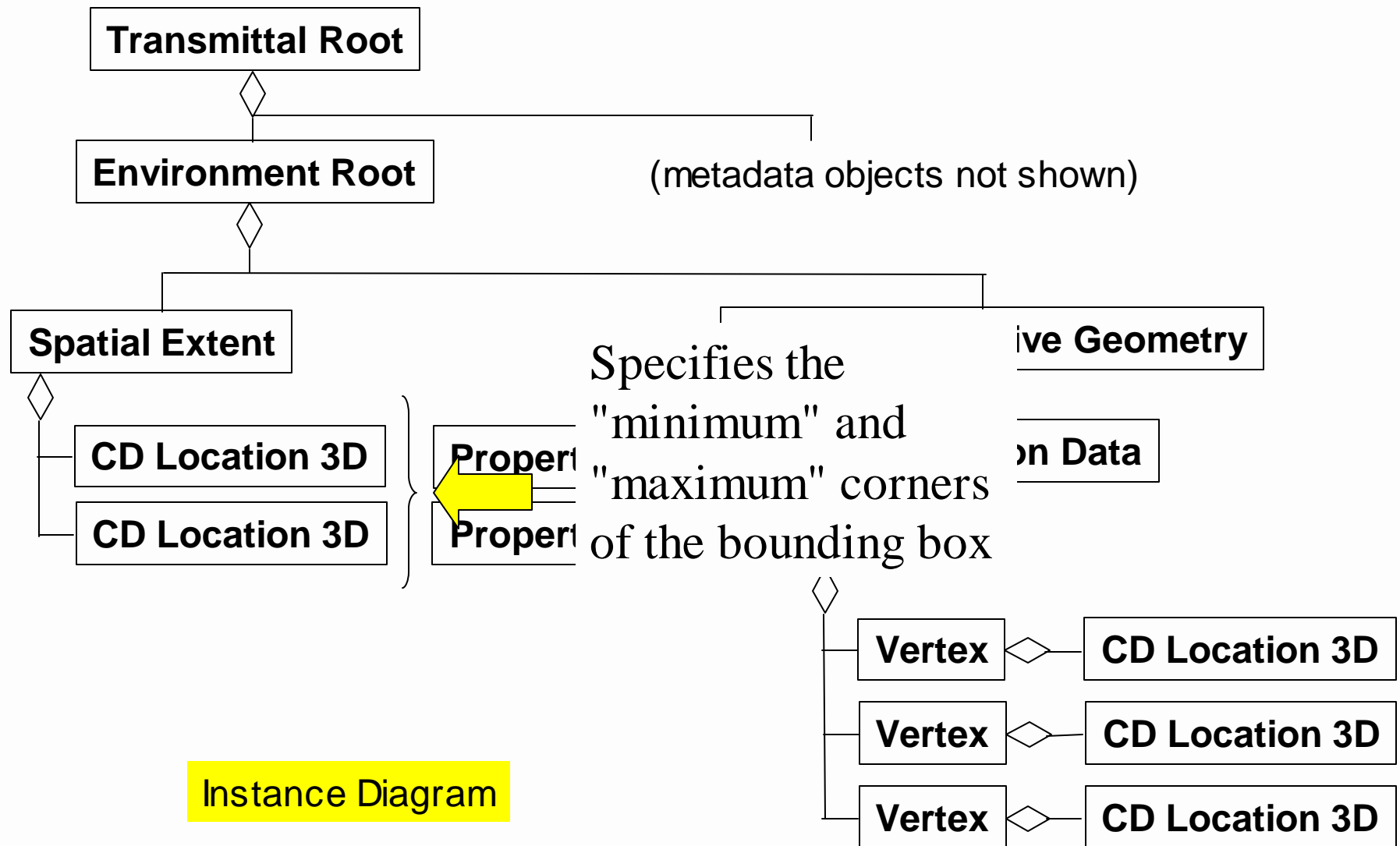
- An is-a relationship between 2 classes indicates that one is a subclass of the other - that any object belonging to the subclass is also a member of the superclass. The triangle, rather than drawing order, indicates which is the superclass.
- All superclasses in the DRM are *abstract* - they exist 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.

Spatial Reference Frames and <Location>

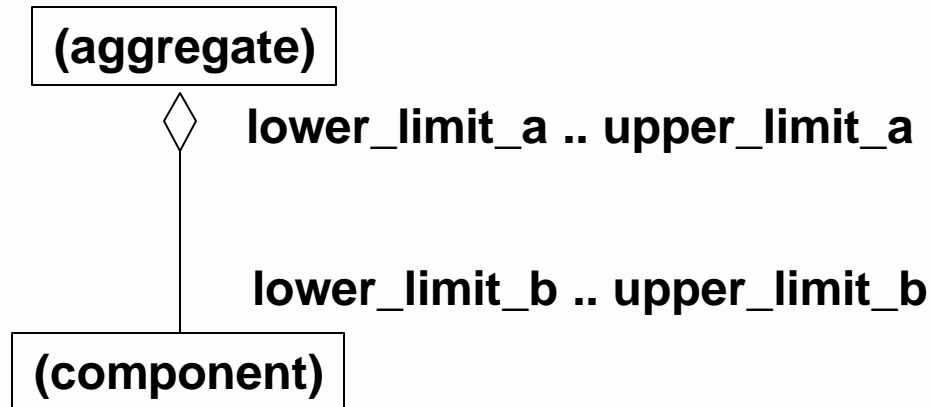


- All relationships in the DRM that need locations are specified with the <Location> class or its abstract subclasses, so all the SRM's spatial reference frames are supported for all such relationships.
- The requirement that a given <Location> shall be valid in the spatial reference frame it appears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

<Spatial Extent>: Specifying a Bounding Box

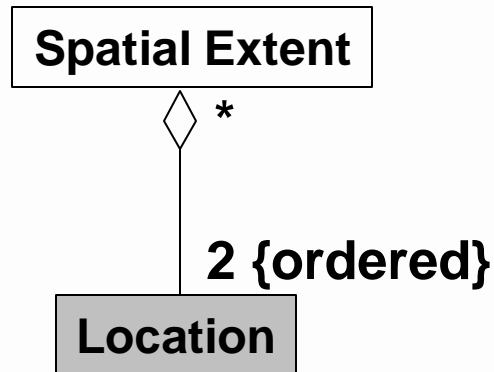


UML Notation Reminder: Multiplicity and Ordering



- 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}.

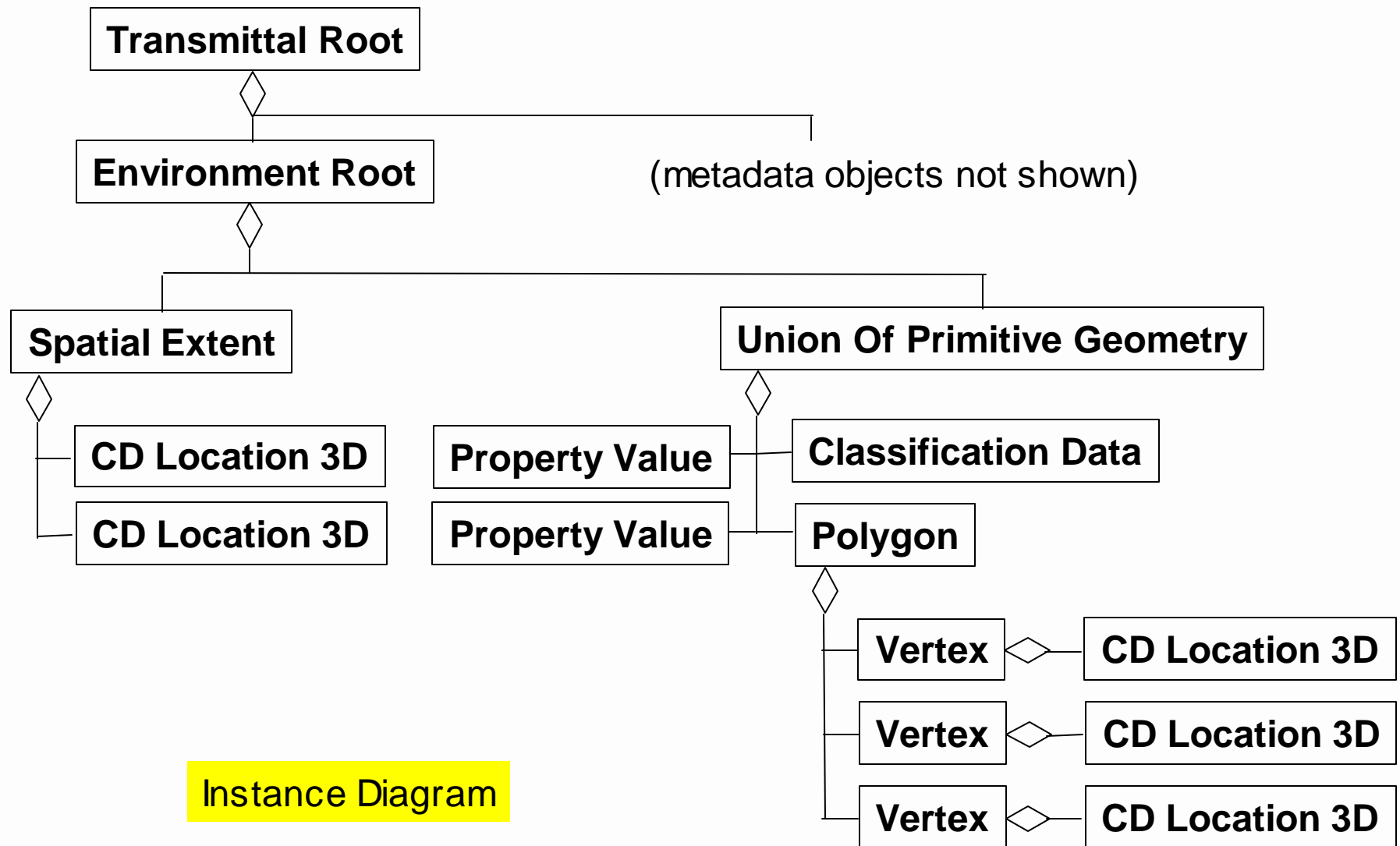
The <Spatial Extent> Class



Detail, Class Diagram Sheet 20

- The class specifies that every <Spatial Extent> instance has 2 ordered <Location> components, and that
 - The first <Location> component is the "minimum" corner of the bounding box - the southwest corner.
 - The second <Location> component is the "maximum" or northeast corner of the bounding box.
- A <Location>, for its part, may be shared by any number of <Spatial Extent> instances.

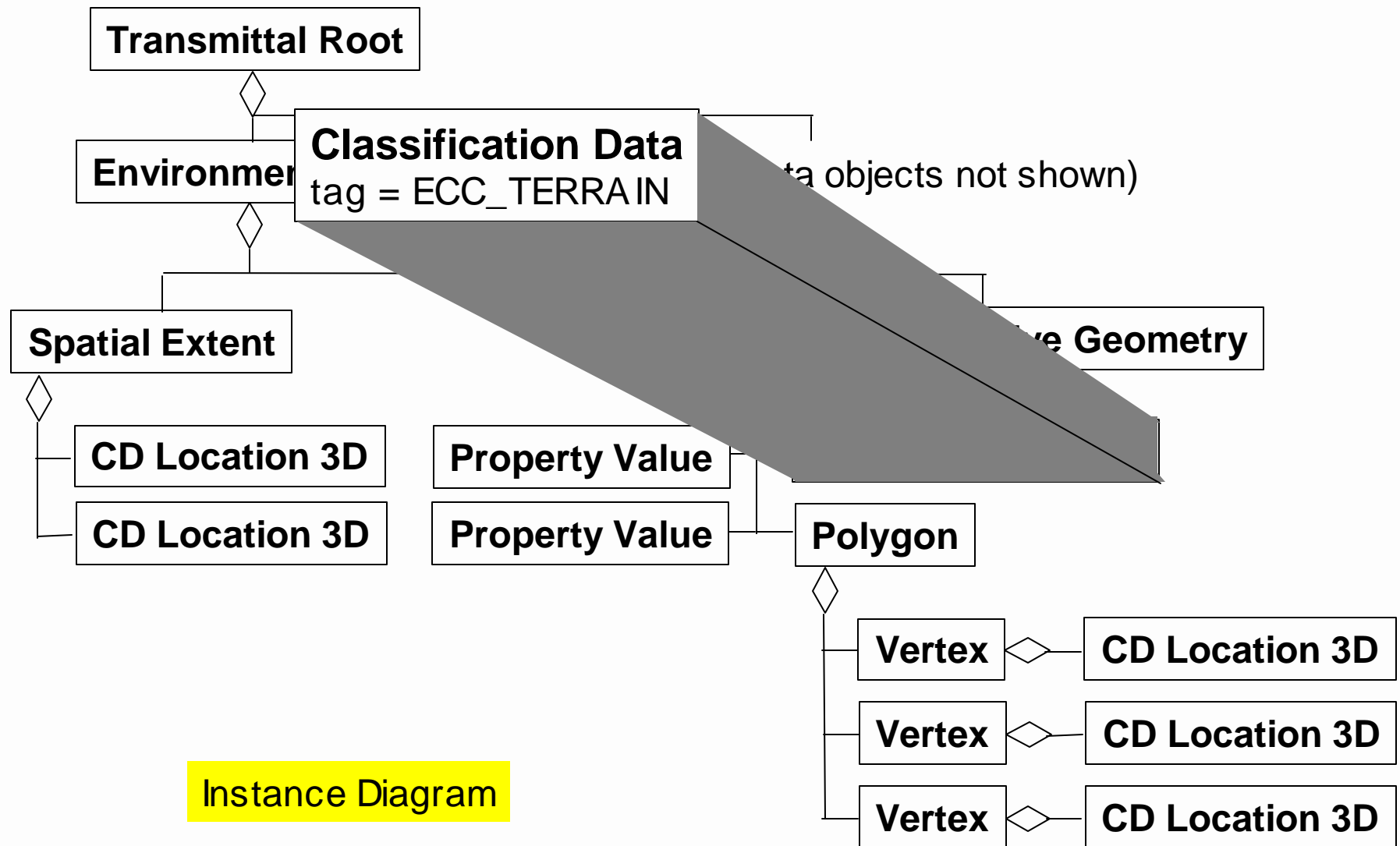
<Environment Root>: The Environment Representation



How <Environment Root> Represents the Environment

- The geometric representation in this example is organized as a <Union Of Primitive Geometry>, where the primitive geometry consists of <Polygon> instances.
- Under the "union" organizing principle, a "bag of stuff" — another level of organization or actual primitives, depending on the organizer's class — is grouped together. "Union" is the simplest kind of organization.
- If all higher-level organization semantics are stripped away, all geometric organizations ultimately boil down to combinations of
 - <Union Of Primitive Geometry>
 - <Property Grid Hook Point> (covered later)

What the Representation Represents



<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.

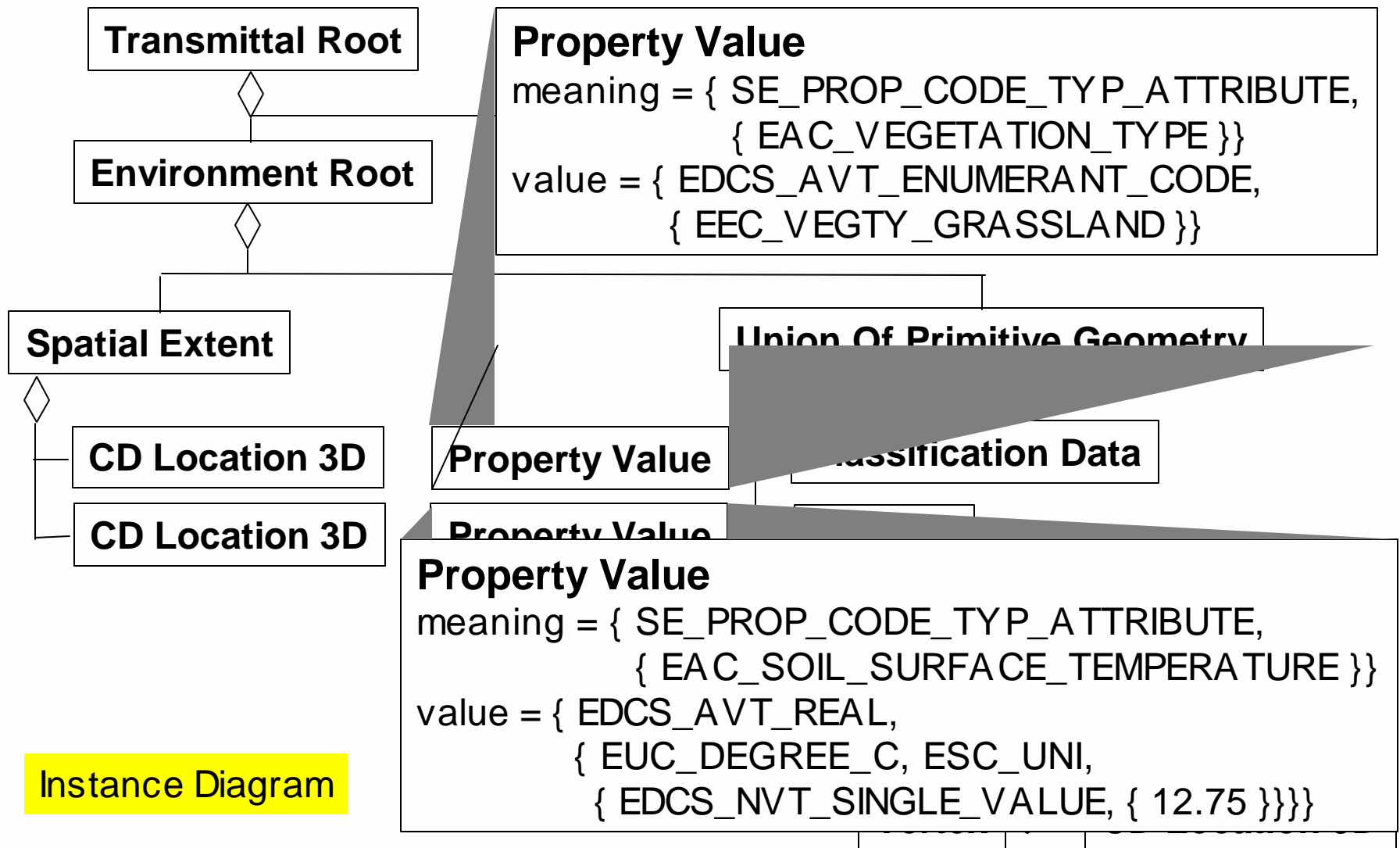
Using EDCS Classification Codes

- The label (such as TERRAIN) 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

Using EDCS Classification Codes

- The label (such as TERRAIN) is just that: a label for a concept.
- Always check the definition in the EDCS Classification Dictionary.
- Examples:
 - A given data provider may want a more specific ECC than ECC_TERRAIN, such as ECC_DRY_LAND or ECC_WATER_BODY_FLOOR.
 - ECC_TRENCH, ECC_MILITARY_TRENCH and ECC_WATER_BODY_FLOOR_TRENCH mean different things.

Some Environmental Properties



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.

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.**

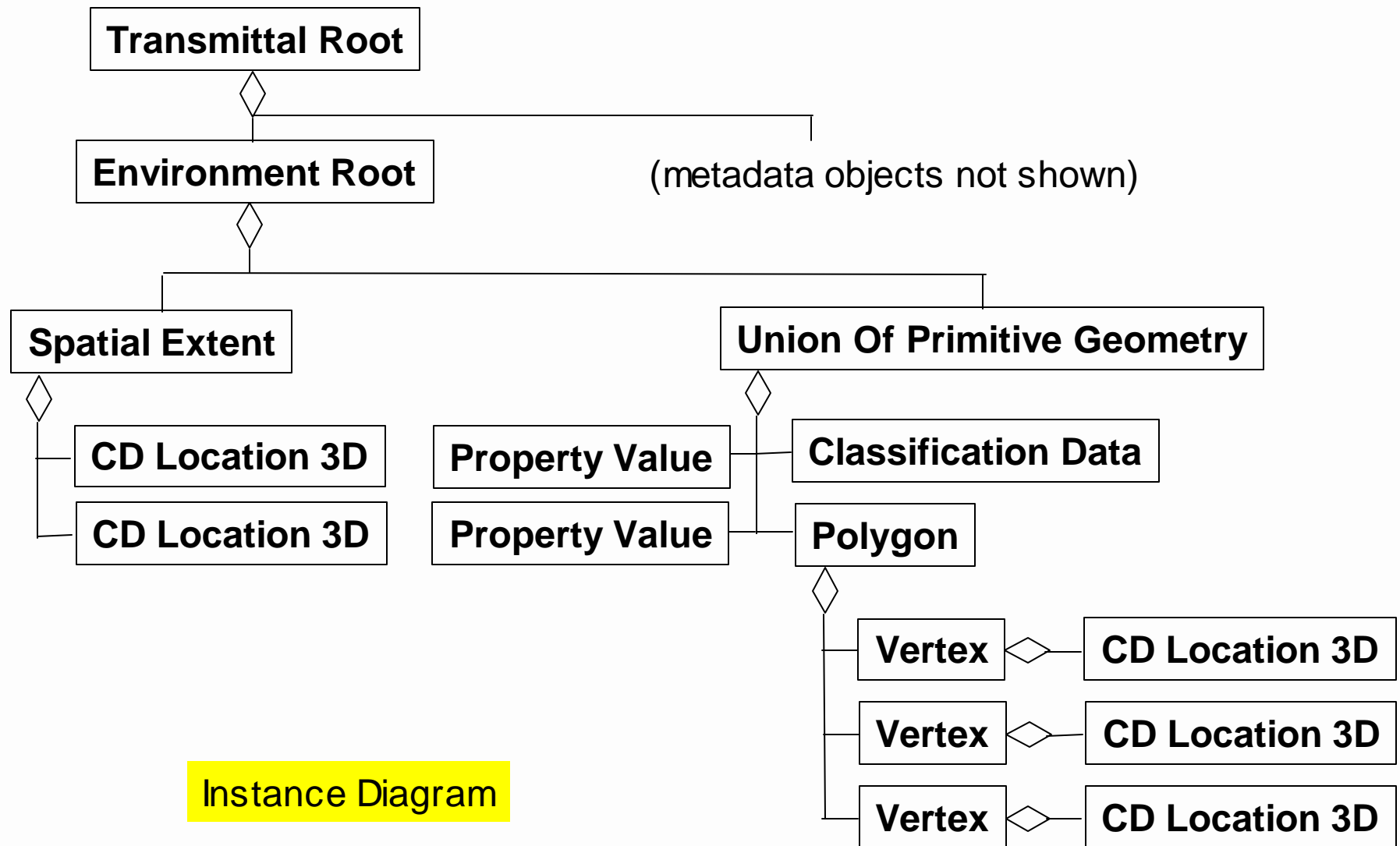
Using EDCS Attribute Codes

- Every EDCS Attribute Code not only has a definition, but is bound to an abstract data type.
- For example:
 - EAC_VEGETATION_TYPE in our example is bound to an enumeration specified in EDCS for that attribute, and a legal EAC_VEGETATION_TYPE value must be an enumerant from that list.
 - EAC_AIR_TEMPERATURE can only take a number as a value.
- Furthermore, if an EDCS Attribute Code is some numeric quantity, it must be specified with an appropriate scaled unit of measure.
- In the DRM, if an EDCS Attribute Code is not a numeric quantity, as for EAC_VEGETATION_TYPE, EUC_UNITLESS and ESC_UNI are used.

Specifying <Property Value> Instances

- In the current example, the <Union Of Primitive Geometry> has two <Property Value> components.
- This is unambiguous, because the <Property Value> instances in this case comply with the <<No Attribute Conflicts>> constraint. That is, they're specifying values for different attributes — EAC_VEGETATION_TYPE and EAC_SOIL_SURFACE_TEMPERATURE — rather than conflicting values for the same attribute.

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 an ECC_TREED_TRACT?
- 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:

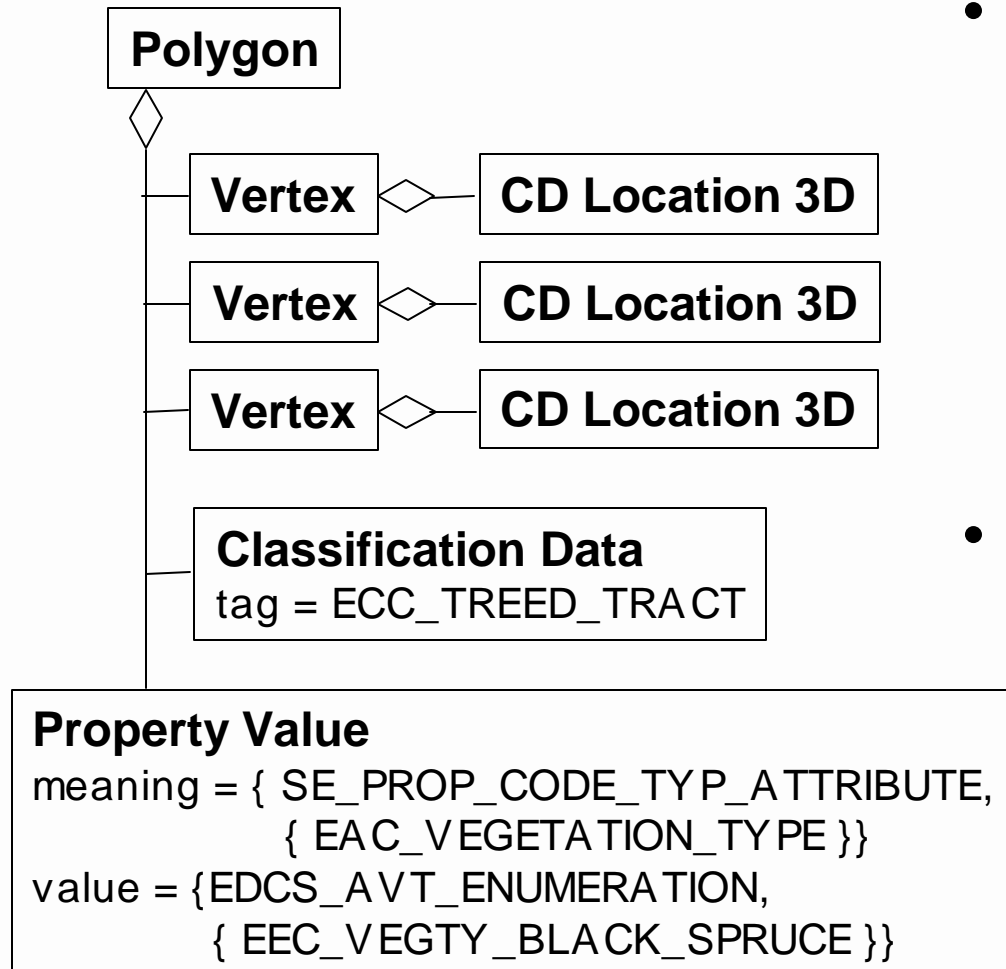
<Classification Data>

<Property Value>

<Colour>

<Image Mapping Function>

Overriding Inherited 'Attribute' Components



- Not only can we unambiguously specify an **ECC_TREED_TRACT** <Polygon> in an **ECC_TERRAIN** <Union Of Primitive Geometry>, but we can override other inherited information as well.
- No conflicting attributes may be specified at the *same* level of the tree, but if a different value for an attribute is specified at a *lower* level, it is considered to override the inherited value.

Instance Diagram

<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>

Final Thoughts on Primitive Geometric Representation

- The principles you have learned for primitive geometric representation provide a foundation for learning the basics of any representation.
- <Union Of Primitive Geometry> can organize instances of any of the subclasses of <Primitive Geometry>, including <Point>, <Line>, <Ellipse>, and <Finite Element Mesh>, but <Polygon> is currently the most commonly used.

<Environment Root> Topics

- We begin by defining the term spatial reference frame, and showing how the DRM organizes spatial data.
- We apply this knowledge, showing the functionality provided by an <Environment Root> to support geometric and feature representations of an environment, as well as presenting the fundamentals of such representations, supported by examples.
 - Fundamentals of primitive geometric representation
 - **Fundamentals of tabular representation**
 - Fundamentals of feature representation

Fundamentals of Tabular Representation

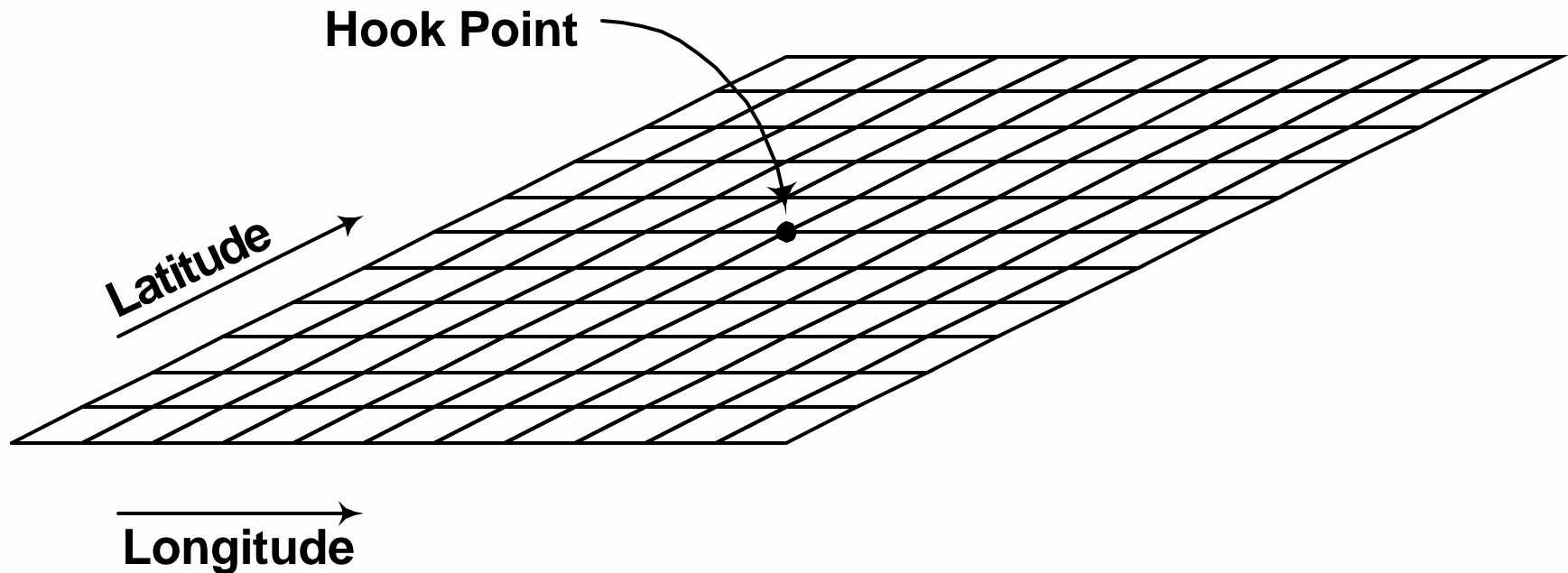
What classes are needed for a tabular, or *gridded*, representation of an environment?

The classes introduced in this section are

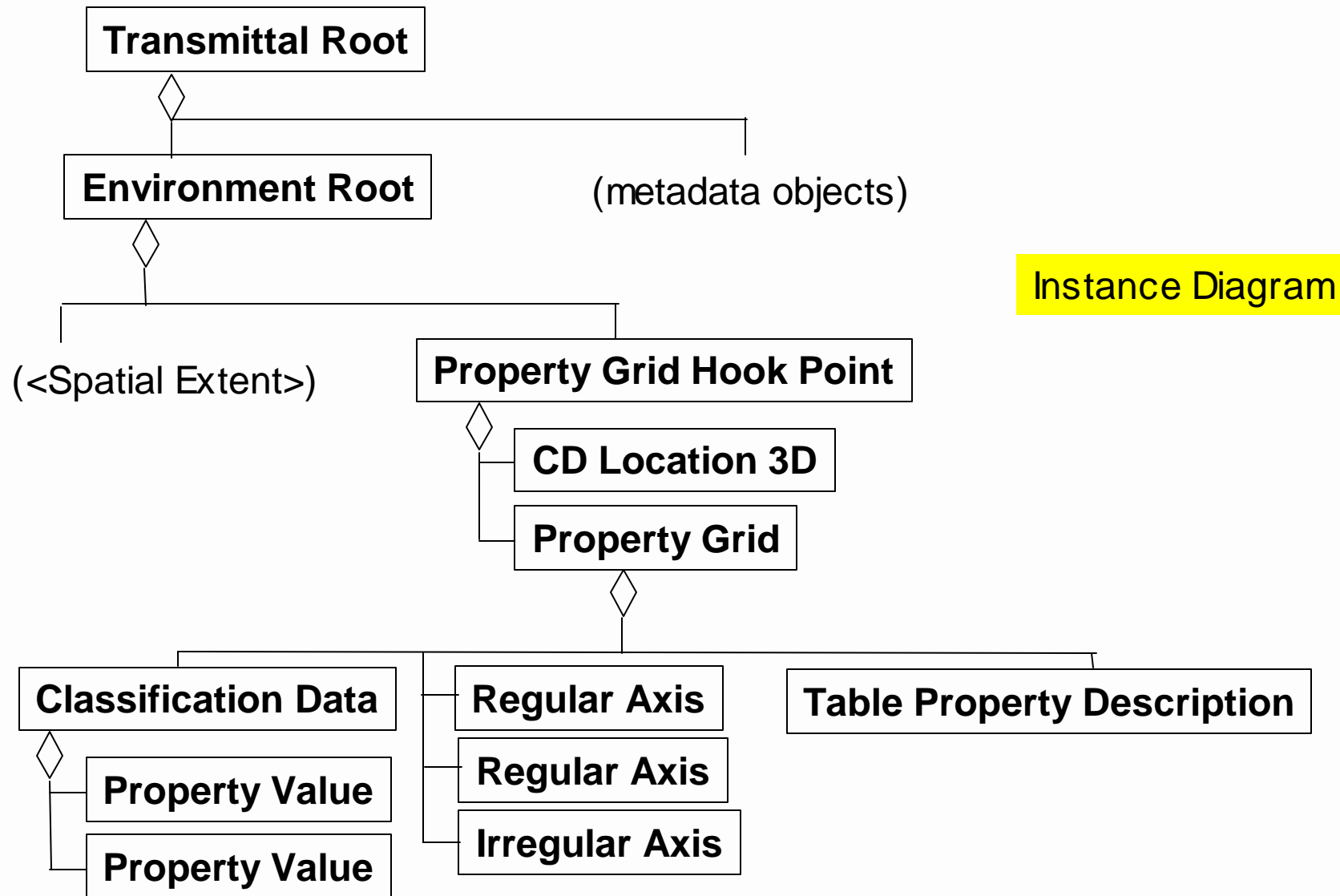
- <Property Grid Hook Point>
- <Property Grid>
- <Regular Axis>
- <Irregular Axis>
- <Table Property Description>
- <Property>

Tabular Representation: <Property Grid>

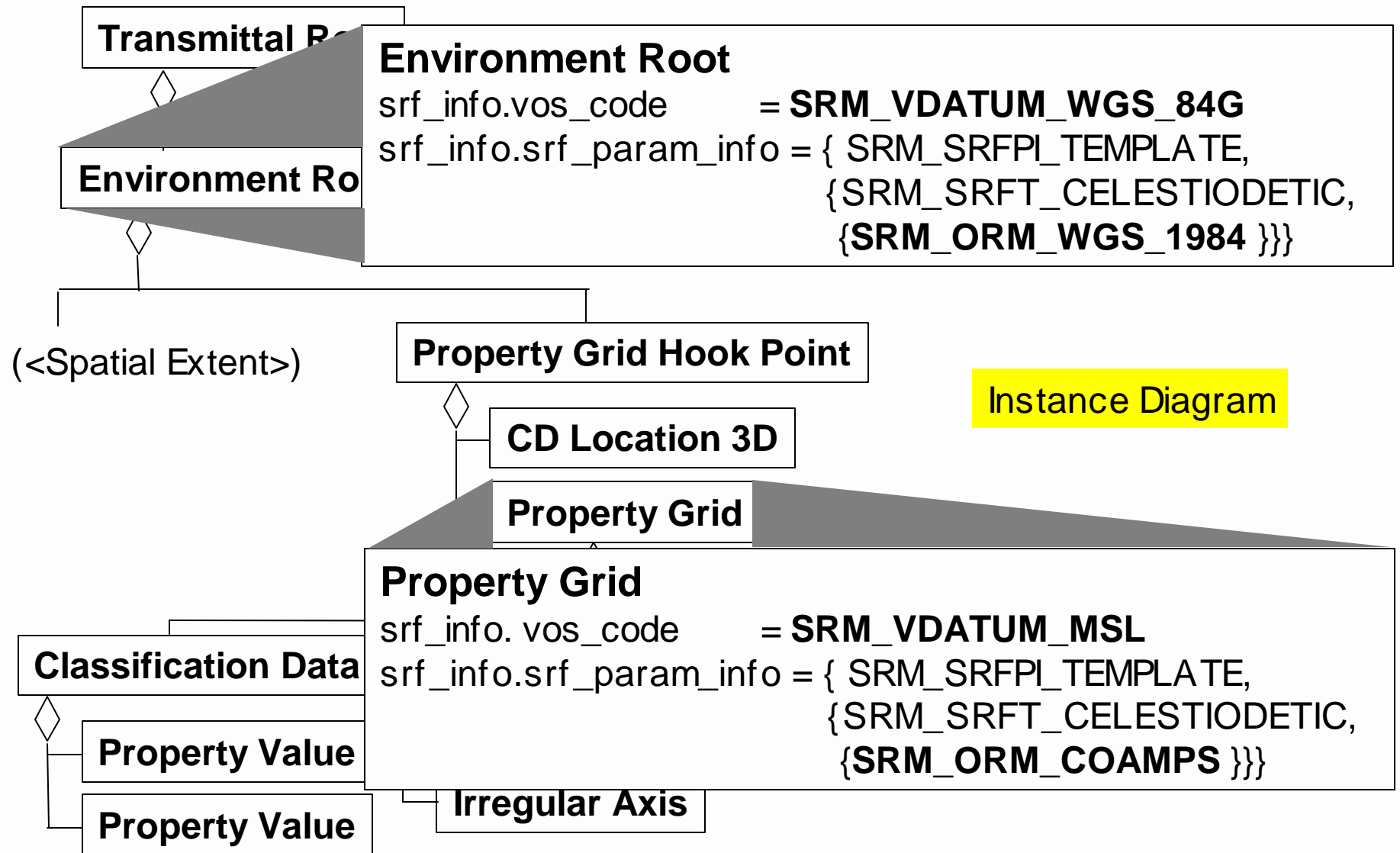
A <Property Grid> instance specifies the values of one or more properties for each cell of a grid, which covers some region.



Tabular Representation: A <Property Grid> Example



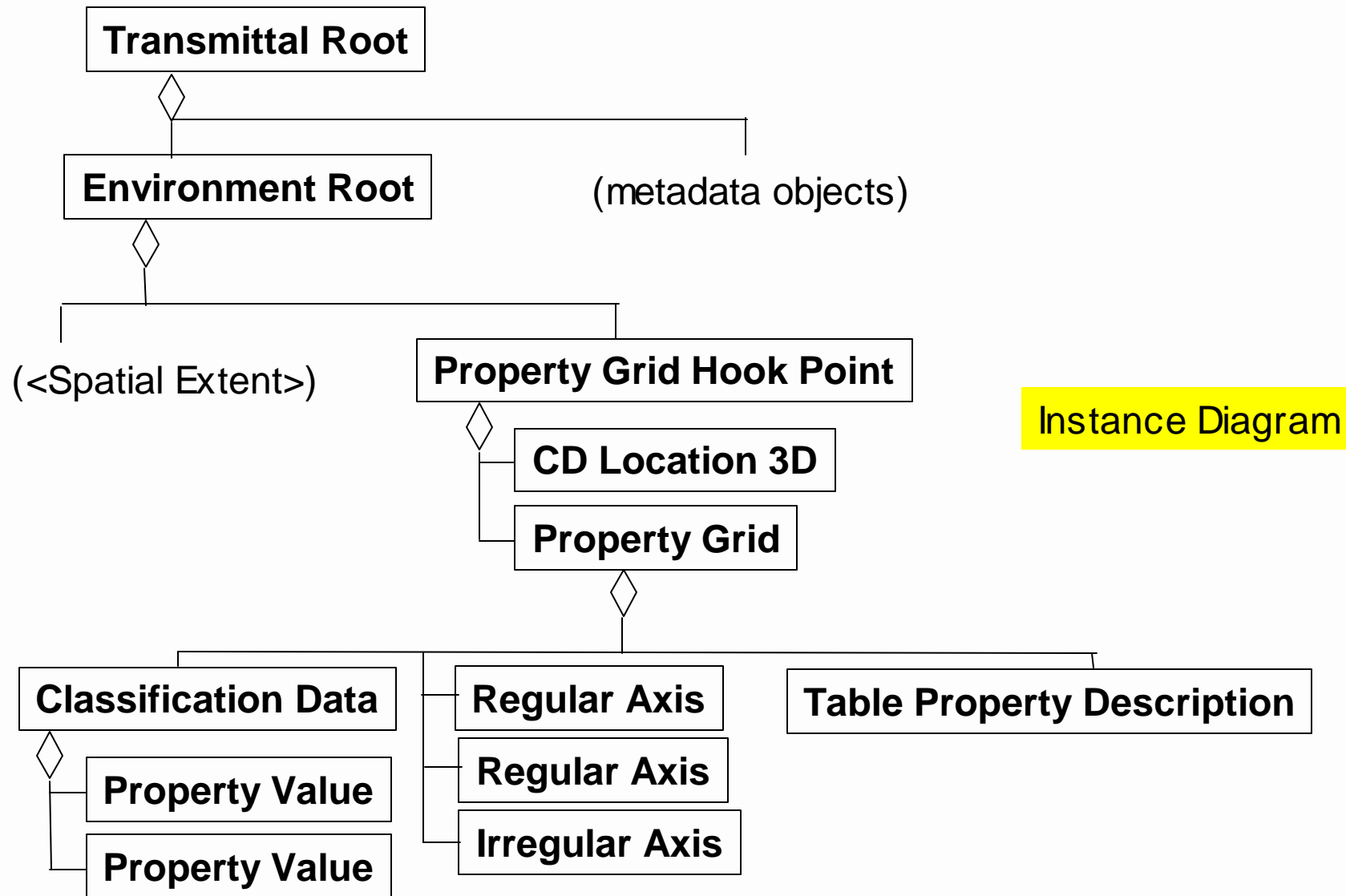
<Property Grid> Has Its Own Spatial Reference Frame



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

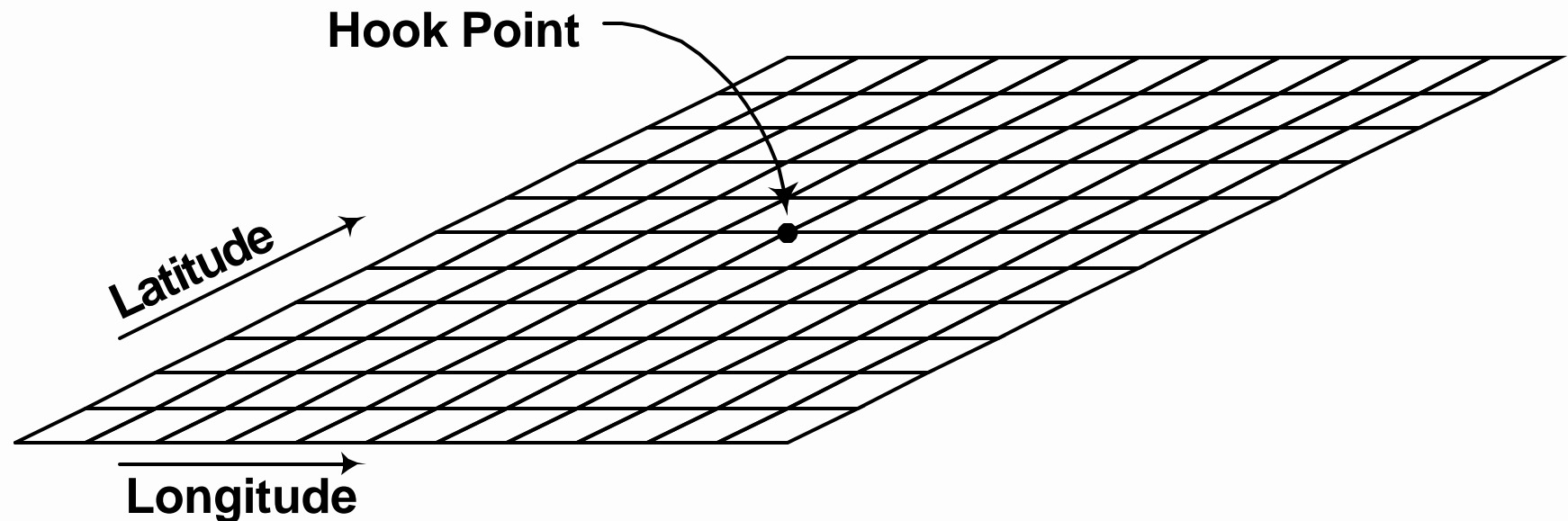


<Property Grid> and Spatial Reference Frame Issues

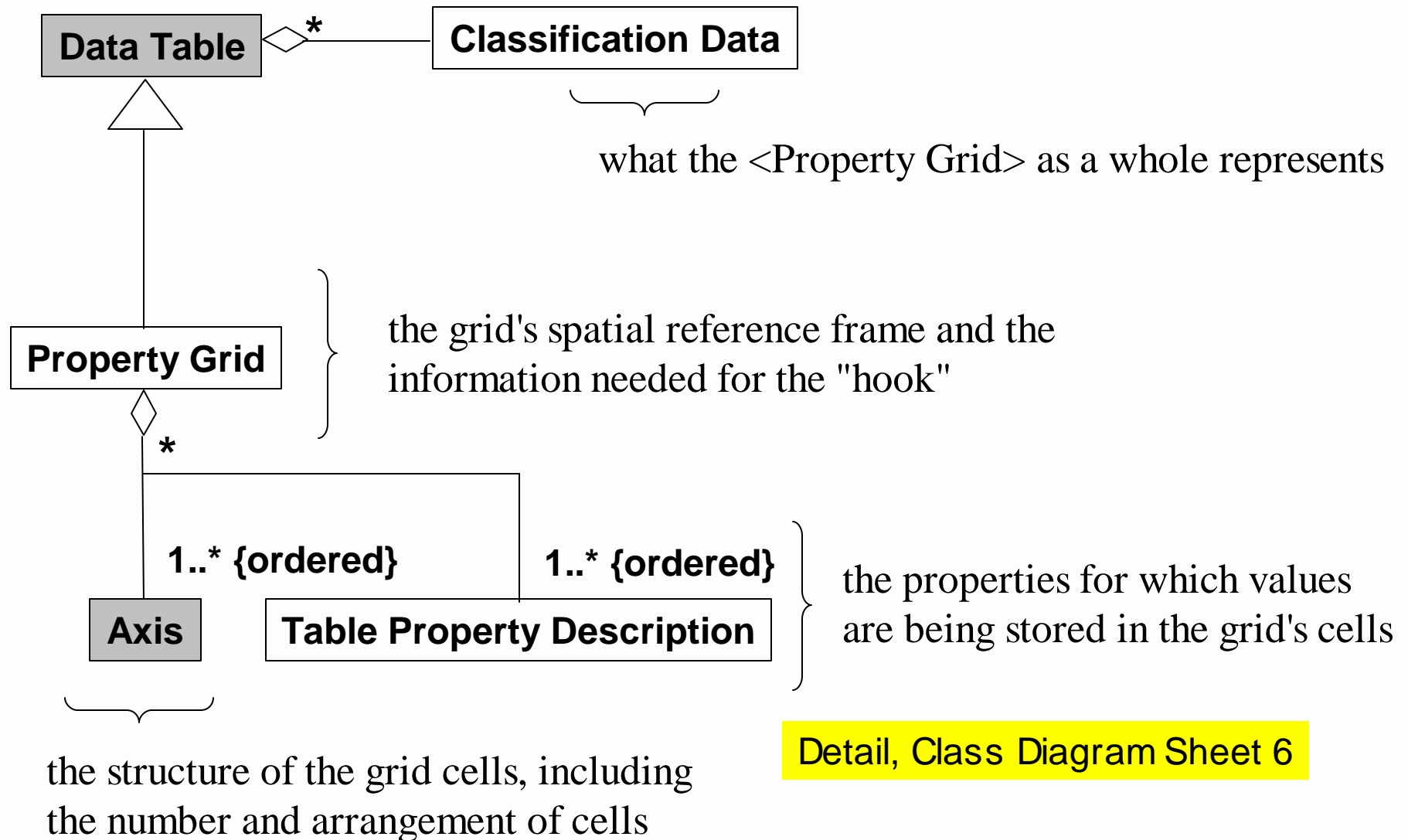
- **A <Property Grid> instance specifies its own SRF primarily to ensure that its "griddedness" is preserved if a consumer asks the API to convert the "native" SRF of part of a transmittal to another SRF.**
 - **During coordinate conversion/transformation between different spatial reference frames, <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>.

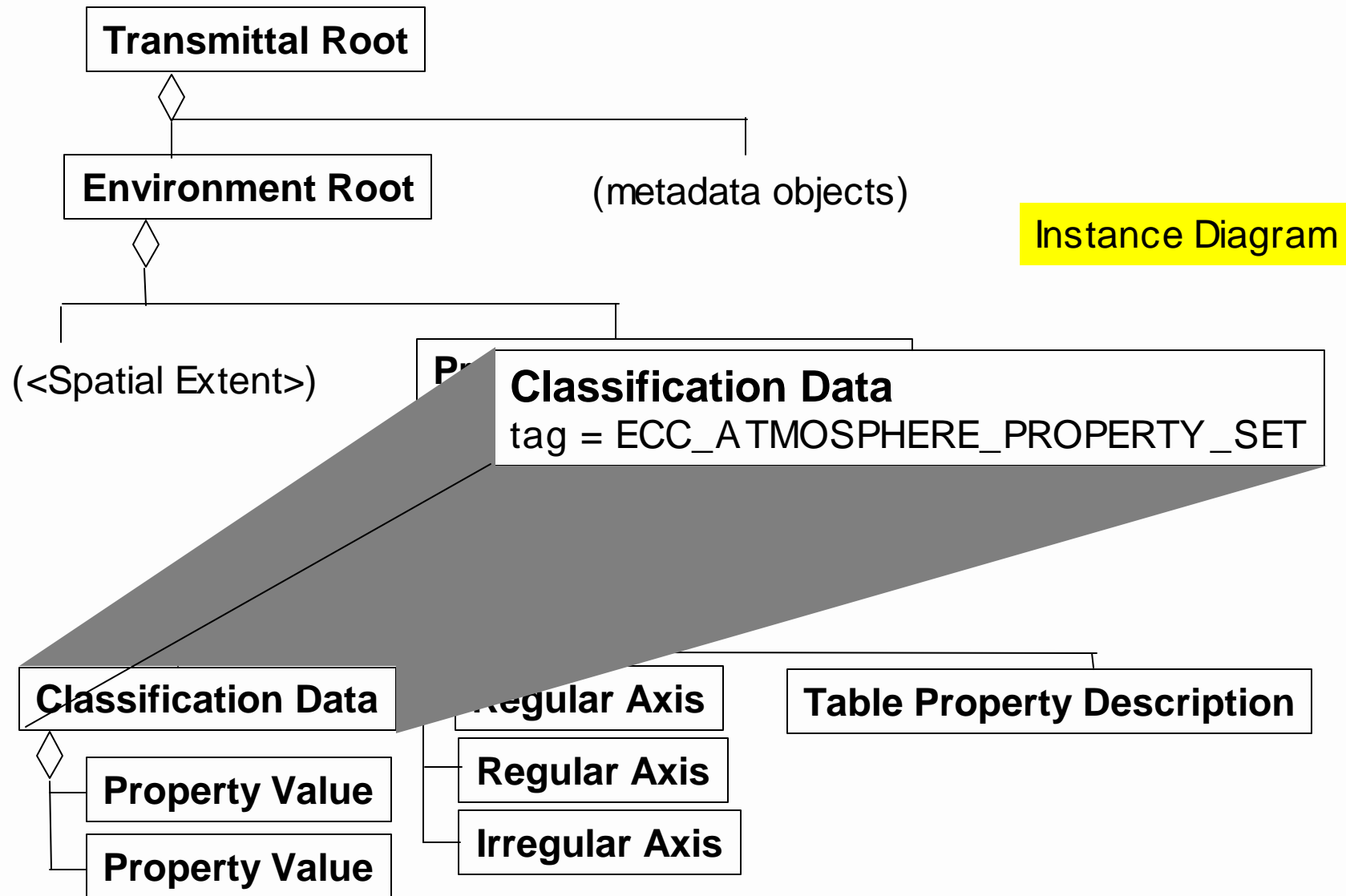


What Every <Property Grid> Instance Must Specify



Detail, Class Diagram Sheet 6

What the <Property Grid> Represents



When <Classification Data> Needs Further Qualification

- **<Classification Data> sometimes requires further elaboration to precisely specify the kind of thing the data provider wishes to represent.**
 - That an ECC_BUILDING functions as a bank.
 - That an ECC_BRIDGE is structurally a drawbridge.
- In this example, we have an **ATMOSPHERE_PROPERTY_SET** that was obtained by analysis and which covers a volume. For atmospheric physicists, that information can be considered part of specifying exactly what kind of data set it is - part of its classification.

What the <Property Grid> Represents, After Elaboration

Instance Diagram

Transmittal Root

Classification Data

tag = ECC_A TMOSPHERE_PROPERTY_SET

Property Value

meaning = { SE_PROP_CODE_TYP_ATTRIBUTE,
 { EAC_PROPERTY_SET_DATA_SOURCE } }
value = { EDCS_AVT_ENUMERATION,
 { EEC_PRPSETDATSRC_ANALYSIS } }

Property Value

meaning = { SE_PROP_CODE_TYP_ATTRIBUTE,
 { EAC_PROPERTY_SET_SPATIAL_DOMAIN } }
value = { EDCS_AVT_ENUMERATION,
 { EEC_PRPSETDATSRC_VOLUME } }

Prope

Tabular Representation: The <Axis> Components

Regular Axis

axis_type = EAC_SPATIAL_ANGULAR_PRIMARY_COORDINATE
value_unit = EUC_DEGREE_ARC
value_scale = ESC_UNI
(other fields)

(<Spatial

Regular Axis

axis_type =
EAC_SPATIAL_ANGULAR_SECONDARY_COORDINATE
value_unit = EUC_DEGREE_ARC
value_scale = ESC_UNI
(other fields)

Classification Data

Property Value

Property Value

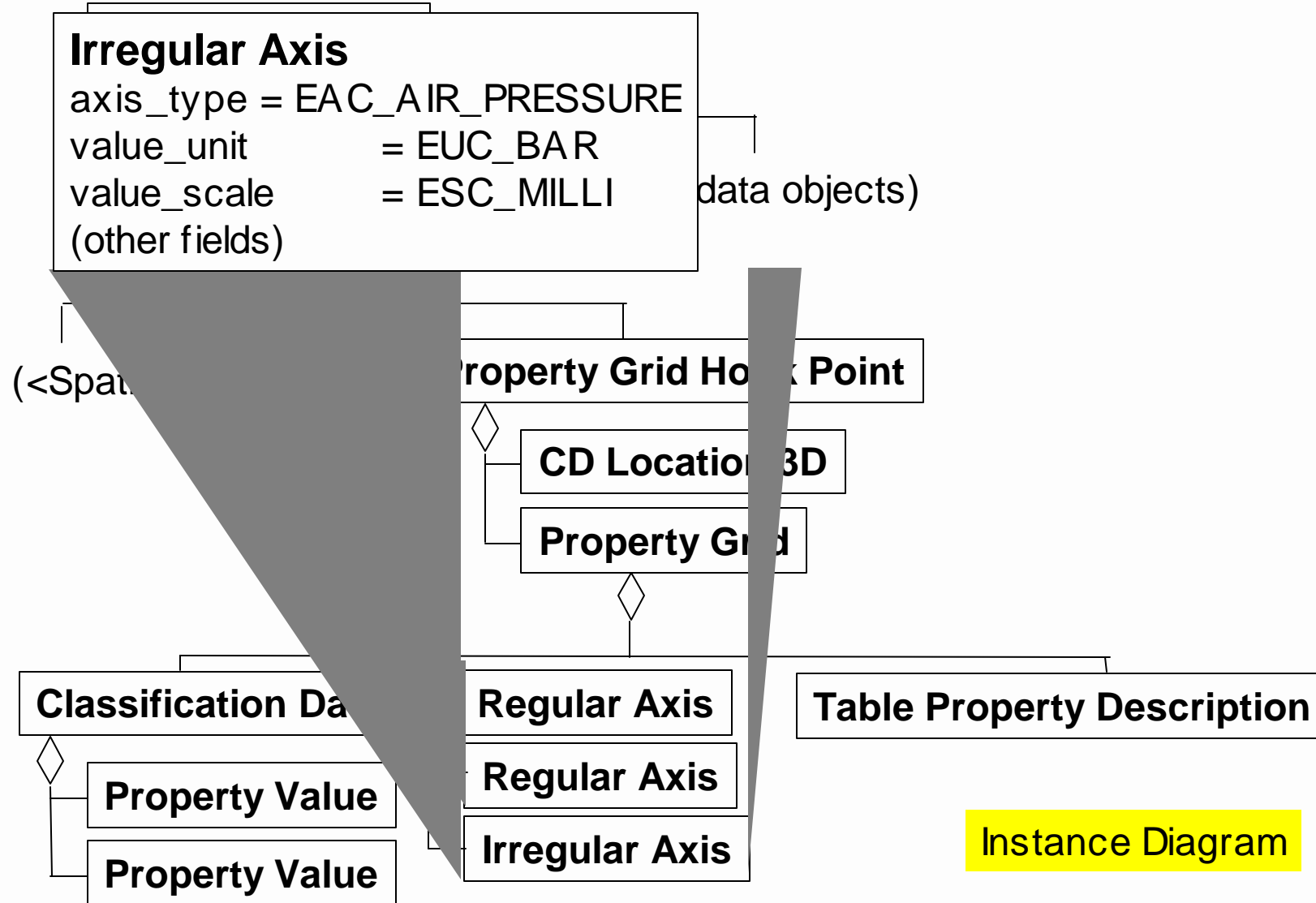
Regular Axis

Regular Axis

Irregular Axis

Instance Diagram

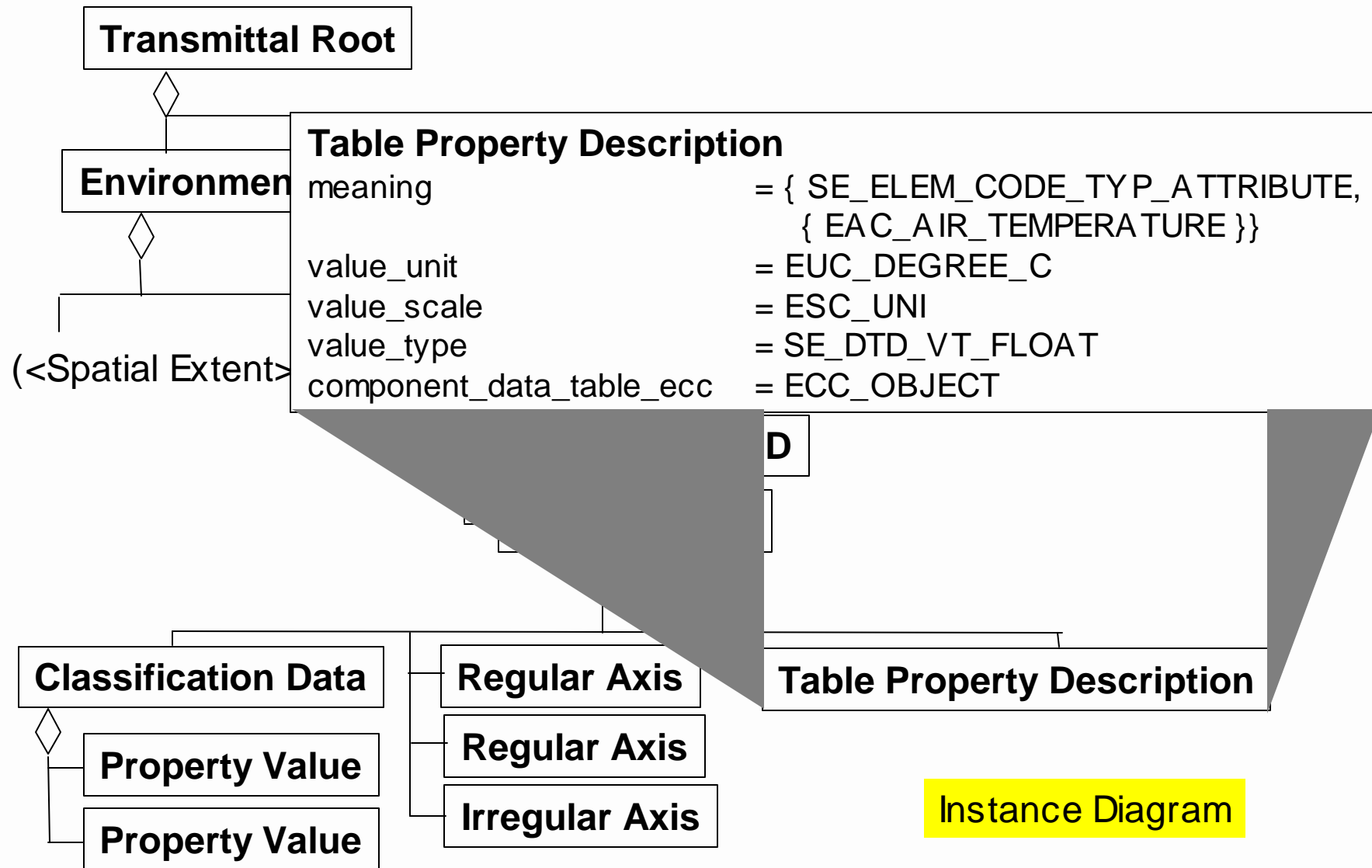
Tabular Representation: The <Axis> Components



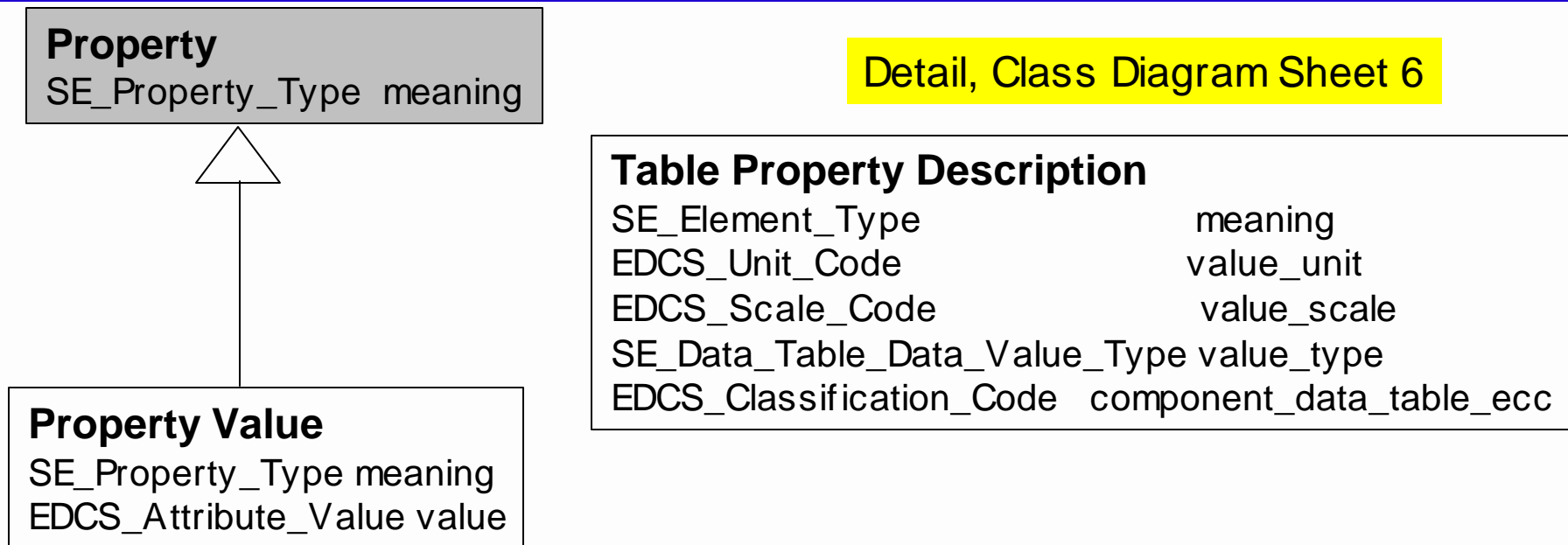
<Regular Axis> and <Irregular Axis>

- 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.
- For a more detailed treatment of <Axis> instances, attend *Advanced Application of the DRM*.

Tabular Representation: A <Property Grid> Example



<Table Property Description>



Detail, Class Diagram Sheet 6

<Table Property Description> has similar functionality to the <Property Value> class, except that

- a TPD instance indicates the data type of the associated cell data rather than containing the values themselves, and
- a TPD instance can indicate extra information when a <Data Table> refers to other <Data Table>s' contents (see *Advanced Application of the DRM* tutorial for details).

Final Thoughts 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>.**

<Environment Root> Topics

- We begin by defining the term spatial reference frame, and showing how the DRM organizes spatial data.
- We apply this knowledge, showing the functionality provided by an <Environment Root> to support geometric and feature representations of an environment, as well as presenting the fundamentals of such representations, supported by examples.
 - Fundamentals of primitive geometric representation
 - Fundamentals of tabular representation
 - **Fundamentals of feature representation**

Fundamentals of Feature Representation

What classes are needed for a feature representation of an environment?

The classes introduced in this section are

- <Union Of Features>**
- <Union Of Feature Topology>**
- <Feature Node>**
- <Feature Edge>**
- <Feature Face>**
- <Point Feature>**
- <Linear Feature>**

Feature Topology

- A <Feature> represents something in the environment so as to abstract away any spatial information that isn't needed to reason about that "thing" in terms of its spatial connectivity.
- A <Feature> does not directly contain spatial information; instead, it is associated with some <Feature Topology> organization.
- The topology of a <Feature> contains the spatial data and further information about its connectivity with other data, while the <Feature> itself supplies such things as <Classification Data> and <Property Value> instances that apply to the entire "thing".

Organization of Topological Data in the DRM

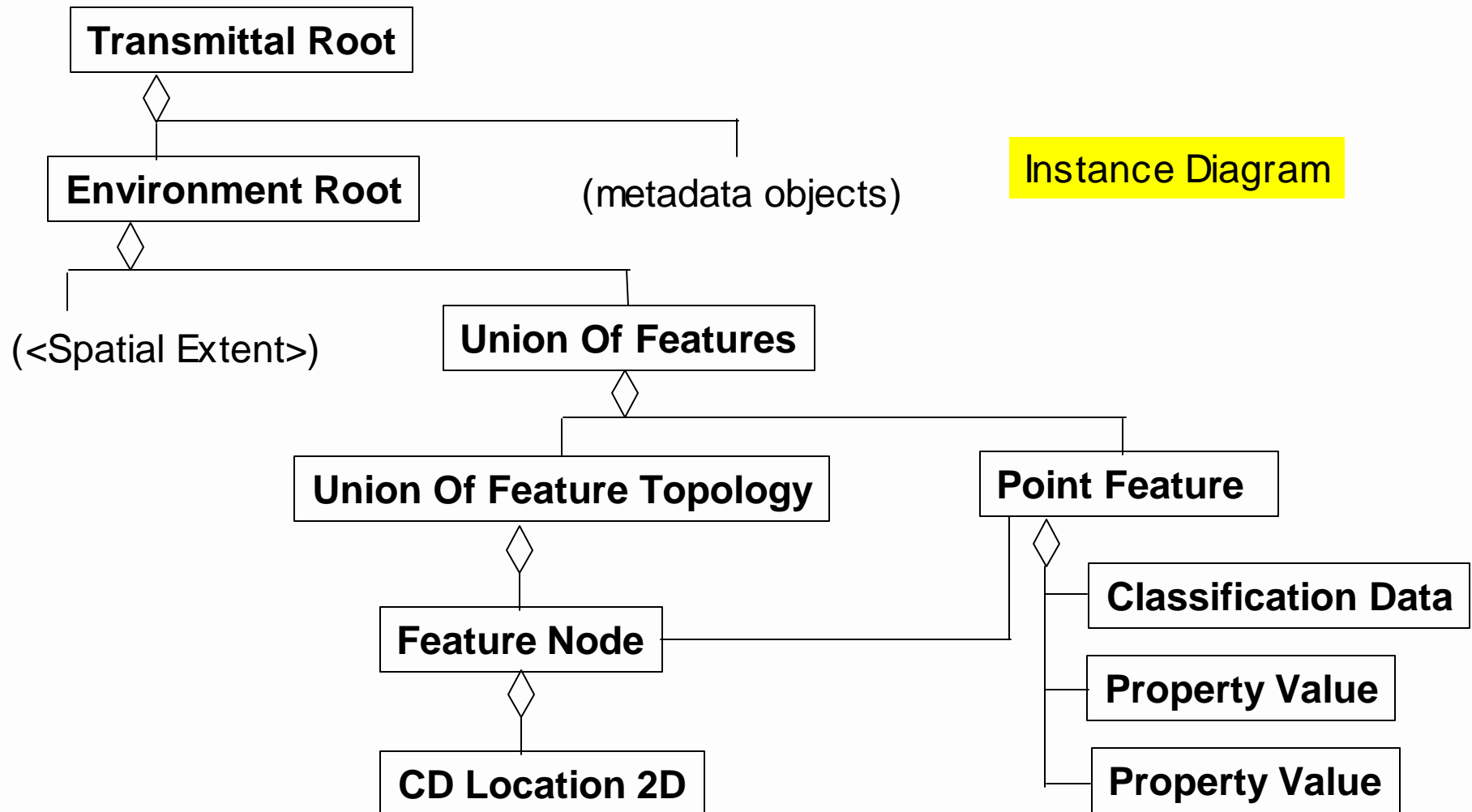
- Topology consists of nodes, edges, and faces - instances of the <Feature Node>, <Feature Edge>, and <Feature Face> classes - and the relationships between instances of those classes.
- The lowest-level organizer for topology primitives is a <Union Of Feature Topology> instance.
- At some level of every feature organization, the corresponding topology organization will appear as a component of a feature organization.

UML Notation Reminder: Association

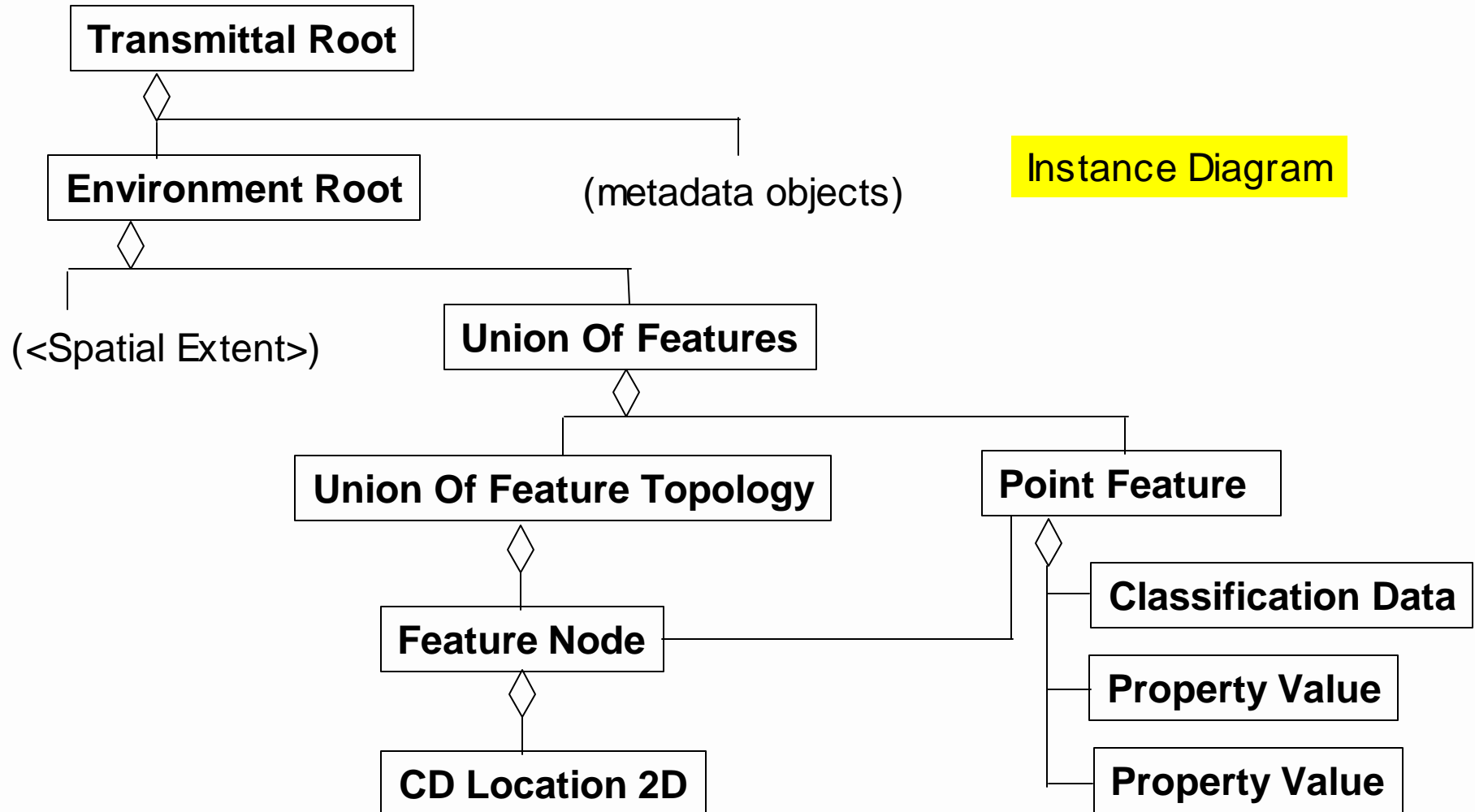


- **An association relationship between classes indicates that instances of those classes can participate in a relationship defined by those classes.**
 - Relating an object to its topology
 - Indicating that two objects are alternate representations of the same "thing"
 - Referencing an object from a <Library>
- **An association can be directional - that is, one participant in the relationship 'knows' that it is participating, but the other doesn't.**

Organizing Primitive Features: An Example



<Environment Root>: The Environment Representation



How <Environment Root> Represents the Environment

- The feature representation in this example is organized as a <Union Of Features>, where one of the features being organized is a <Point Feature> instance.
- Under the "union" organizing principle, a <Union Of Features> is just a "bag" of <Feature> instances.
- If all higher-level organization semantics are stripped away, all feature organizations ultimately boil down to <Union Of Features> instances that organize <Primitive Feature> instances.
- Similarly, all feature topology organizations ultimately consist of <Union Of Feature Topology> instances.

Feature and Topology Topics

- **<Feature Node>**
 - How it can be used in constructing **<Point Feature>** instances, and in creating edges.
- **<Feature Edge>**
 - How it can be used in constructing **<Linear Feature>** instances, and in creating faces.
- **<Feature Face>**
 - How it can be used in constructing **<Areal Feature>** instances.
- **<Union Of Feature Topology>**
 - How it is used in indicating the quality of the topological data it organizes.

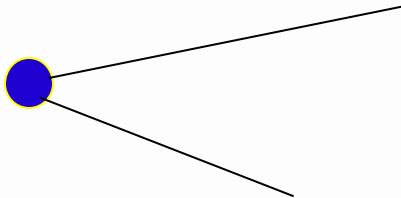
<Feature Node>

Feature Node

0..1

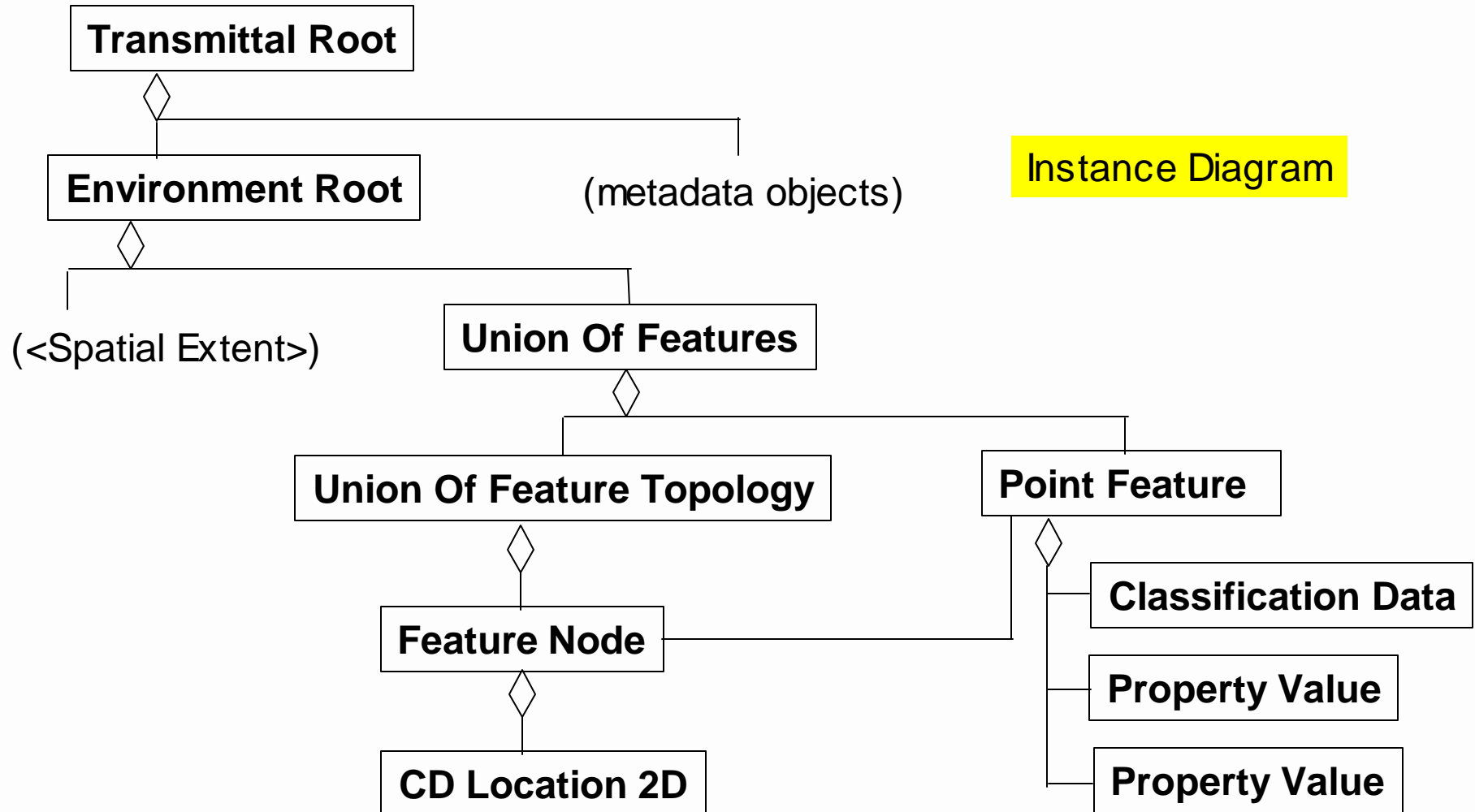
Location

Detail,
Class Diagram
Sheet 12

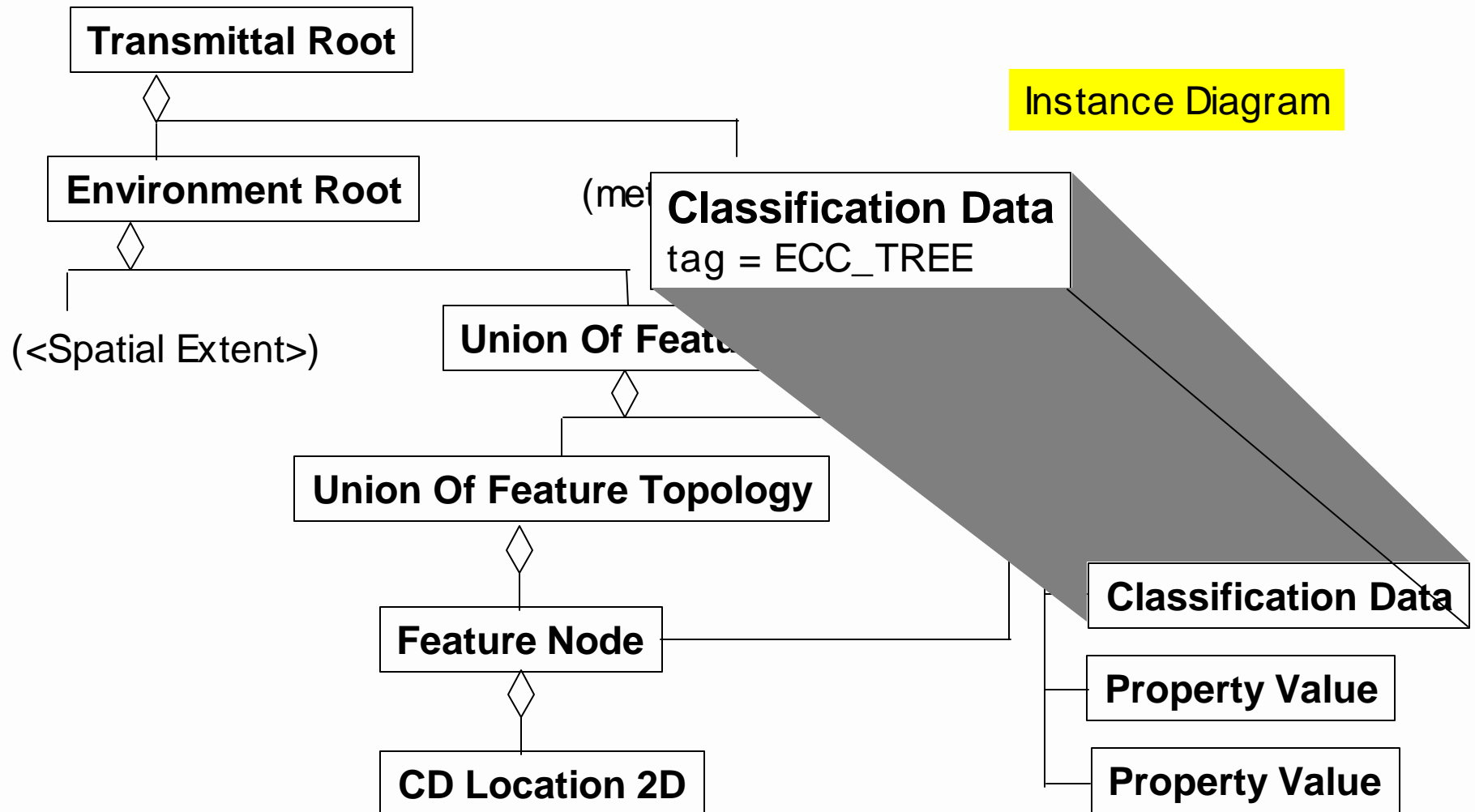


- A <Feature Node> specifies a topologically significant <Location>, which may be either 2D or 3D.
- A <Feature Node> specifies
 - the <Location> of some "thing" in the environment, or
 - the endpoint of some <Feature Edge>, or
 - both
- When a <Feature Node> indicates the presence of some "thing" in the environment, a <Point Feature> will be specified for that <Feature Node>.

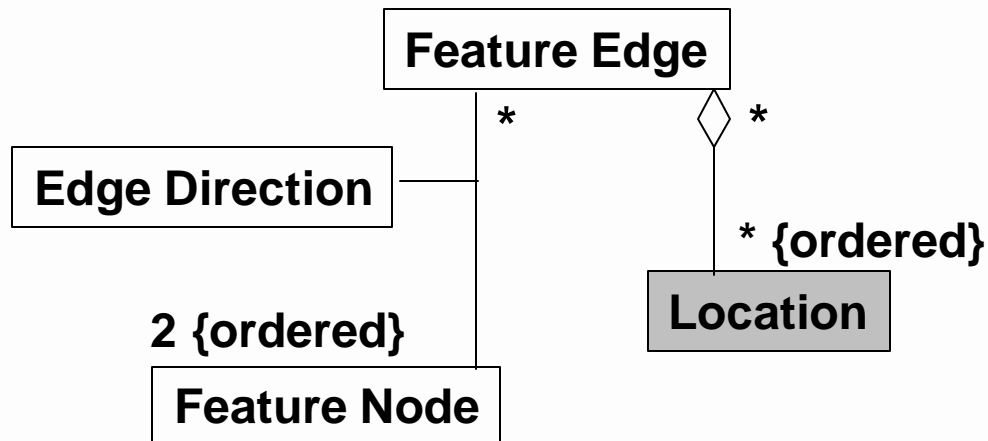
Organizing Primitive Features: <Point Feature> Example



Organizing Primitive Features: <Point Feature> Example

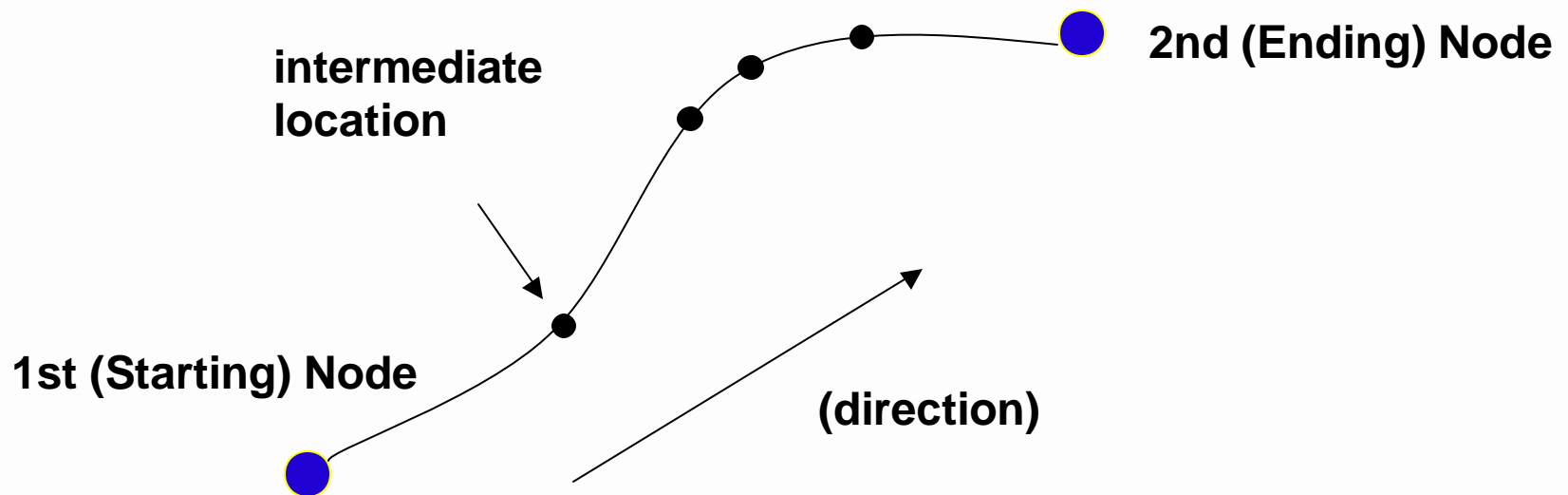


<Feature Edge>



A <Feature Edge> specifies a path in space, which may be either 2D or 3D, by specifying an edge connecting two <Feature Node> associates. If the path isn't a straight line, intermediate <Location>s are specified as components.

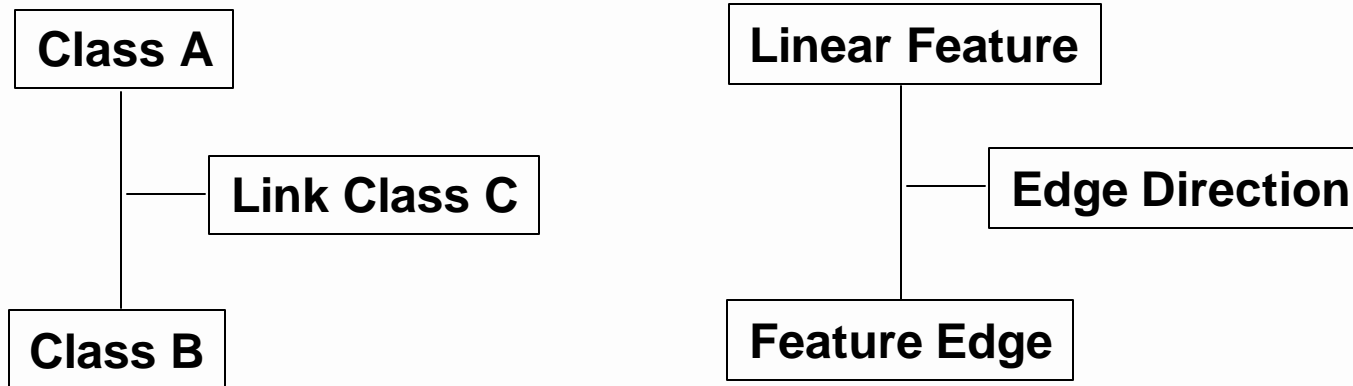
Detail, Class Diagram Sheet 12



Uses of <Feature Edge>

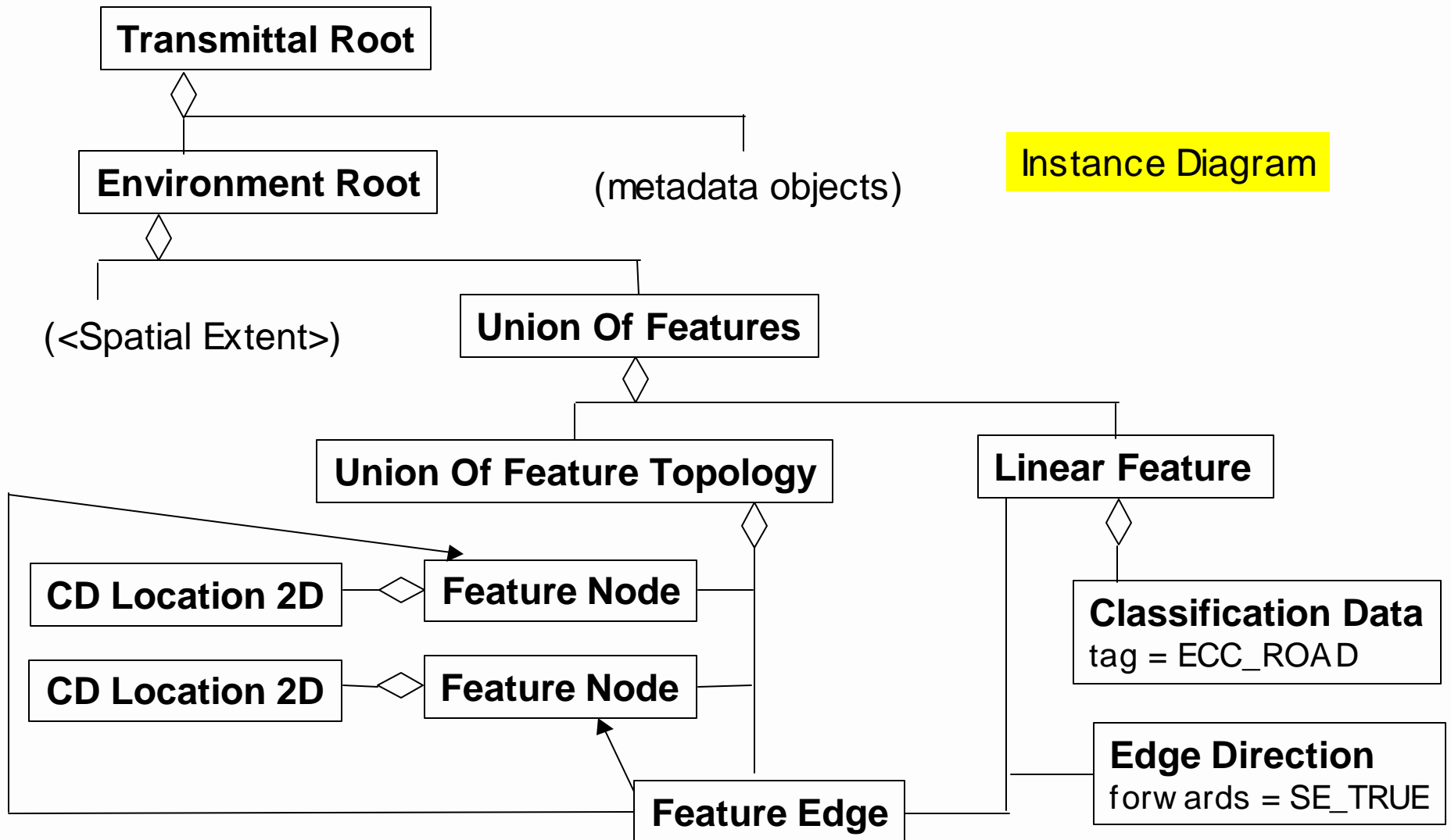
- **A <Feature Edge> specifies**
 - part of the topology of some "linear thing" in the environment, or
 - part of a boundary of some <Feature Face>, or
 - both
- **A <Feature Edge> is therefore either**
 - part of a <Linear Feature>
 - part of a <Feature Face Ring>
 - or both

UML Notation Reminder: Link Classes



- Some relationships require additional information as part of the relationship itself, rather than as part of either class outside that relationship. Such relationships are defined with additional *link objects*.
- The dashed line, rather than drawing order, graphically indicates the presence of a link object on an instance relationship or link class on a formal relationship.

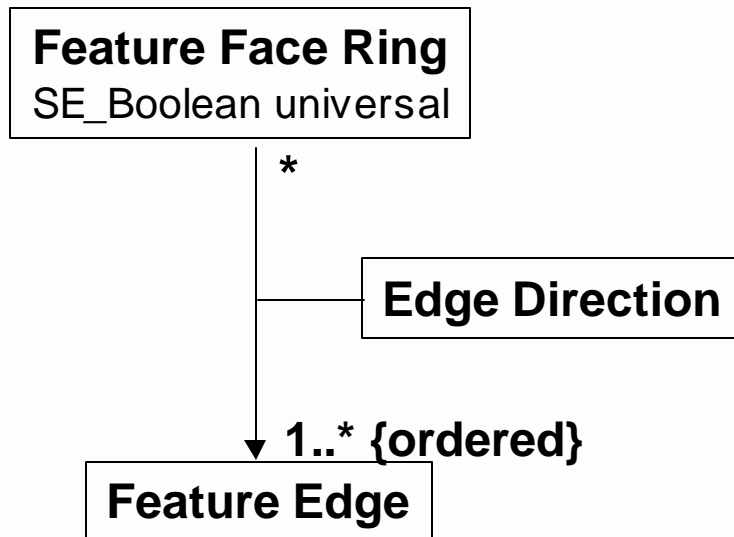
Organizing Primitive Features: <Linear Feature> Example



<Linear Feature> and <Edge Direction>

- A <Linear Feature>'s topology is specified by an ordered sequence of <Feature Edge> instances.
- Each such <Feature Edge> is related to the <Linear Feature> by an <Edge Direction>, indicating whether the <Feature Edge> is to be interpreted forwards (start to end) to join its endpoints with the previous and next edges in sequence.

<Feature Face Ring> and <Feature Face>



Detail, Class Diagram Sheet 12

- A <Feature Face Ring> is used to specify a boundary of a <Feature Face>, and is specified as a sequence of <Feature Edge> instances laid end to end.
- A regular <Feature Face> instance (universal = SE_FALSE) treats its first <Feature Face Ring> as its external boundary. All other <Feature Face Ring> components are *internal* boundaries, representing "holes" in the <Feature Face>.

<Areal Feature> and <Feature Face>

- An <Areal Feature>'s topology is represented by an associated <Feature Face>, which is usually a regular <Feature Face>, through a <Face Direction> link object.
- The <Face Direction> is necessary because in topology involving 3D <Feature Face> instances, an <Areal Feature> may involve only one side of a <Feature Face>.
- An <Areal Feature> may represent such environmental objects as
 - the "footprint" of a building
 - a lake
 - a forest

Topology Organization and Topology Level

- A <Union Of Feature Topology>'s `feature_topology_level` field provides an indication of the kind of topology objects that are present, and the quality of the topological data.
- Generally speaking, the higher the topology level, the less work will be required for the consumer to determine connectivity relationships between objects.

Topology Level

- **Level 0**
 - Topological data has been provided, but at least 2 <Feature Node> instances are located at the same position.
- **Level 1**
 - All <Feature Node> instances have unique <Location> coordinates.
- **Level 2**
 - <Feature Edge> instances intersect only at <Feature Node>s.
- **Level 3**
 - Regular <Feature Face>s touch only at common boundary edges.
 - Every <Feature Face> indicates which <Feature Node> instances are located within its boundaries.
- **And so on...**

Final Thoughts on Feature Representation

- We have gone over only the fundamentals of features and topology in the DRM. The DRM's topology mechanism can specify more detailed connectivity information.
- A structurally similar mechanism, geometry topology, exists in the DRM, but serves the somewhat different function of explicitly capturing adjacency and containment relationships for geometric primitives rather than necessarily things in the environment.
- For further information on features and topology, attend the terrain representation portion of the *Advanced Application of the DRM* tutorial.

Basic Organization Summary

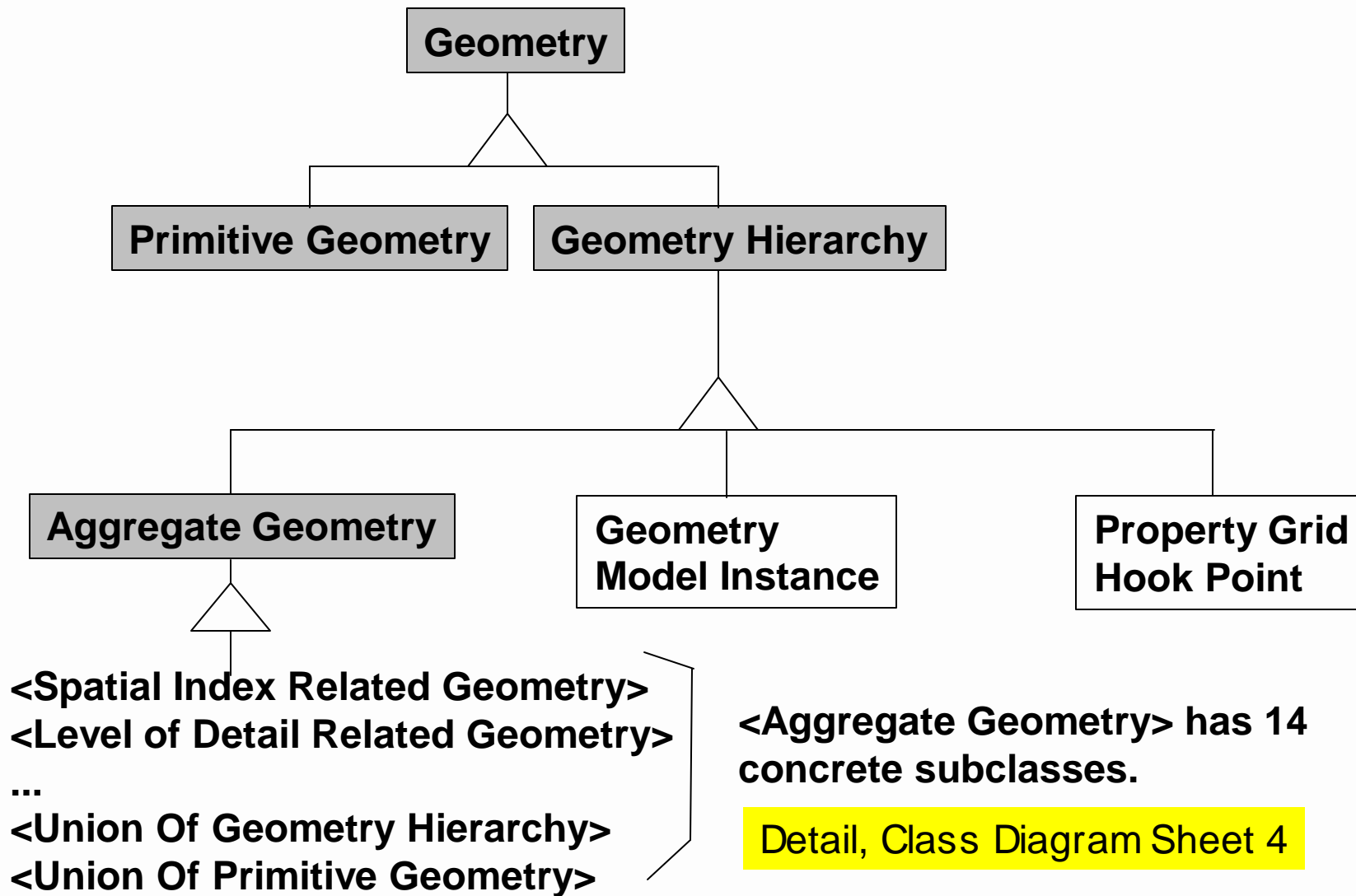
- All geometric representation ultimately consists of organizations of <Union Of Primitive Geometry> and <Property Grid Hook Point> instances.
- All feature representation ultimately consists of organizations of <Union Of Features>.

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.

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.

<Aggregate Geometry>



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.**

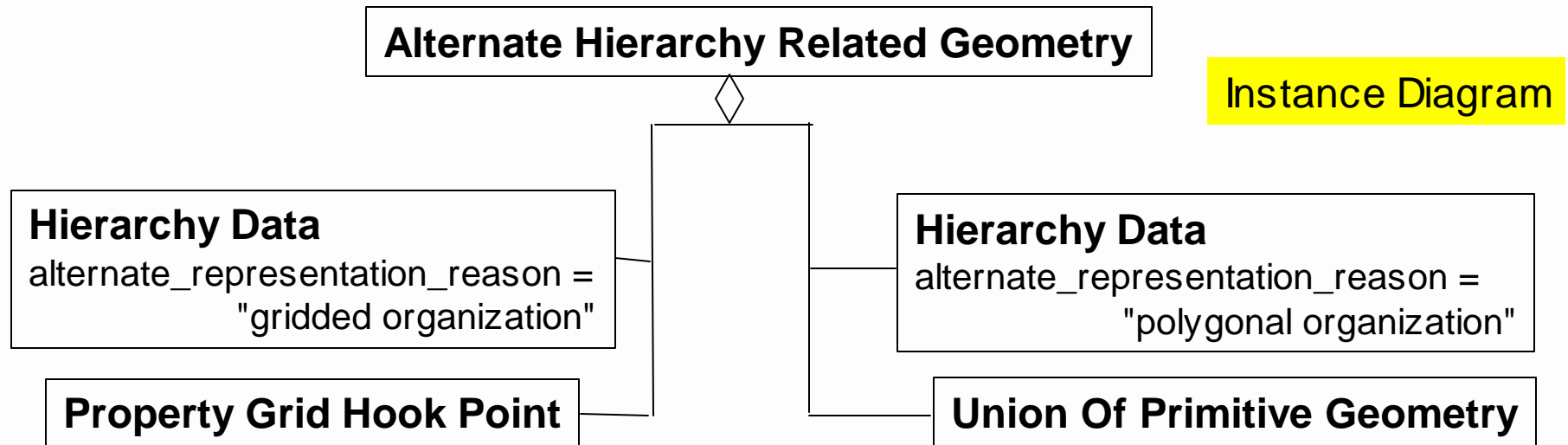
Organizing Principle: Union

- This is embodied by
 - <Union Of Features>
 - <Union Of Primitive Geometry>
 - <Union Of Geometry Hierarchy>
 - <Union Of Feature Topology>
 - <Union Of Geometry Topology>
- These organizers serve to either organize instances of primitives (<Point Feature>, <Polygon>, <Feature Node>) or of other organizers that need to be grouped together but for which no other organizer is more suitable.

Organizing Principle: Alternate Hierarchy

- **Consider two geometric representations of the same data set, such as**
 - **an elevation grid and polygons derived from that grid, or**
 - **two polygonal representations where one set of polygons have been clipped along grid boundary lines but the other hasn't.**
- **To unambiguously indicate that two or more geometric organizations represent the same underlying data, they can be organized as separate branches of an <Alternate Hierarchy Related Geometry>. The <Hierarchy Data> associated with each branch indicates the reason for that particular alternate representation.**

Example: <Alternate Hierarchy Related Geometry>



Alternate Hierarchy Related aggregates are most often found at the top level of the hierarchy - that is, as components of an <Environment Root> - but they can be used elsewhere in the hierarchy as needed.

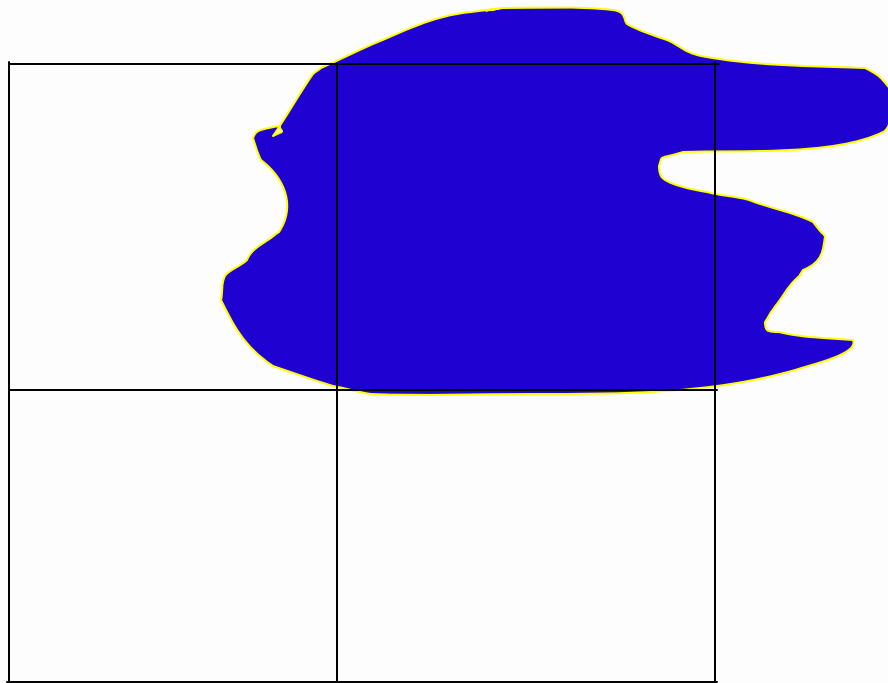
Organizing Principle: Spatial Index

- **A Spatial Index Related aggregate specifies**
 - the number of rows and columns (3 and 3 here)
 - row spacing
 - column spacing
 - spacing units
 - lower left corner of index
- **Each link object specifies which particular (row, column) that component represents.**

<Spatial Index Related Geometry>
(split into 9 cells)

Organizing Principles: Quadrant

- A Quadrant Related aggregate specifies a region being divided into quadrants.
- Each branch of the aggregation specifies the data for a separate quadrant, where the corresponding <Quadrant Data> link object identifies the quadrant.



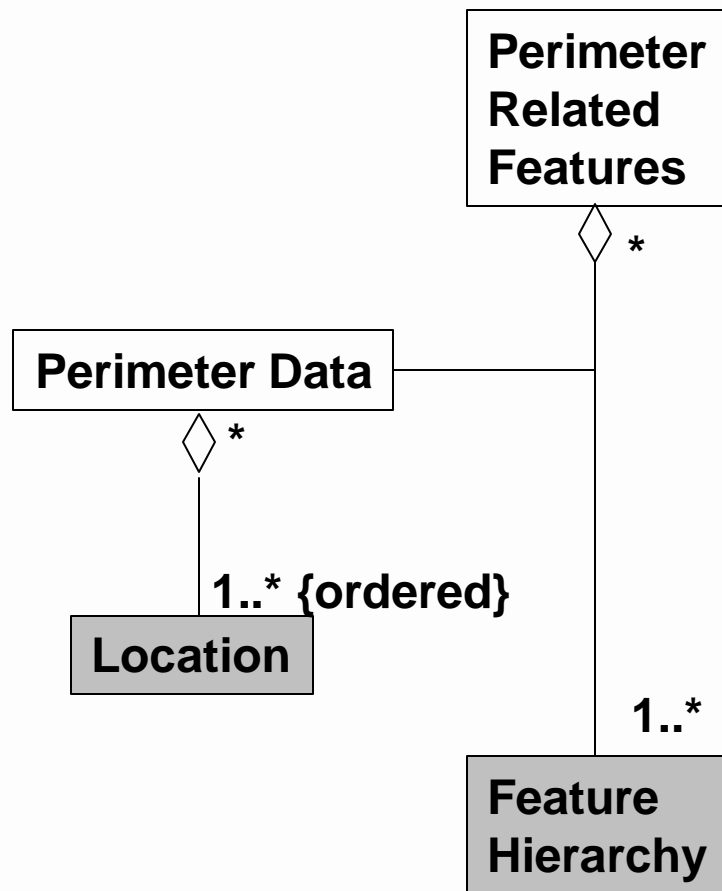
Quadrant Related Organizations

- A Quadrant Related organization is not required to specify data for all 4 quadrants - that is, it may have fewer than 4 branches. In this example, the northeast quadrant is underwater, so a <Quadrant Related Features> concerned with dry land terrain might have no data for that quadrant.
- Note that a Quadrant Related organization is not required to be a "pure" quad tree in the classic sense - its components may or may not be Quadrant Related organizations.

Organizing Principles: Octant

- **An Octant Related aggregation is similar to a Quadrant Related aggregation, but divides a volume into octants rather than quadrants.**
- **Consequently, an <Octant Related Geometry> or <Octant Related Features> would be used for data organized according to volume, such as data describing a volume of atmosphere or of water.**

Organizing Principle: Perimeter

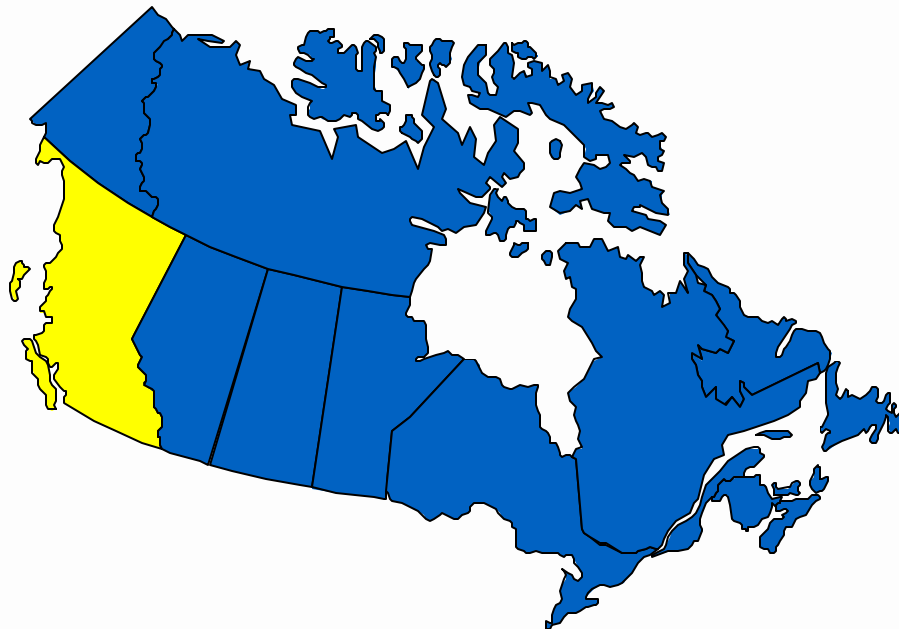


- A Perimeter Related organization is essentially a spatial index organization which uses irregularly shaped partitions, each of which is specified by a <Perimeter Data> link object.
- Each <Perimeter Data> link object specifies the perimeter of the region described by the component object on the corresponding branch.

Detail, Class Diagram Sheet 8

Example:

<Perimeter Related Features>

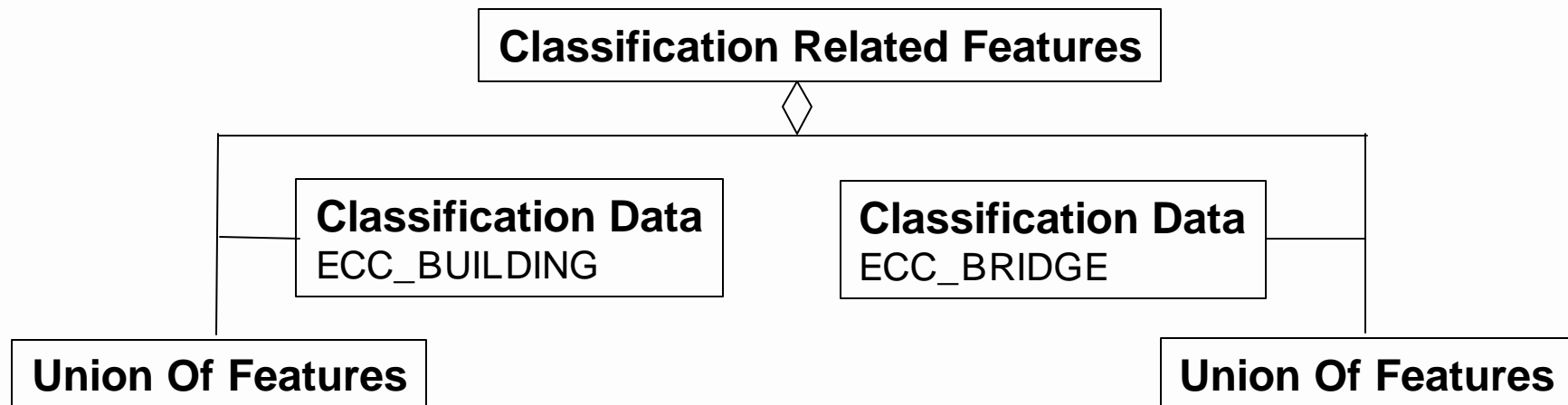


- A <Perimeter Related Features> organization is often suitable for organizing features according to political boundaries.
- If a <Perimeter Related Features> has its `strict_organizing_principle` field set to true, the features in each branch are broken into segments at boundary lines. Otherwise, features may cross the boundary of the branch to which they are assigned.

Organizing Principle: Classification

- A Classification Related aggregate organizes data according to its EDCS Classification.
- In this example, the BUILDING features are in one branch while the BRIDGE features are in the other. The features under a particular branch inherit that branch's <Classification Data>.

Instance Diagram



Organizing Principle: Level Of Detail

A Level Of Detail Related aggregate organizes different representations of the same underlying data that differ in level of detail. It may capture any of several different varieties of the general "level of detail" concept, including:

- Distance - Each branch represents data as seen by an observer at some specified distance range from a given point.**
- Volume - Each branch represents data as seen by an observer inside (or outside) a volume specified for that branch.**
- Spatial Resolution - Each branch specifies data at a different resolution.**
- Map Scale - Each branch specifies data at a different map scale.**

Organizing Principle: Continuous Level Of Detail

- **Continuous Level Of Detail, unlike the more general Level Of Detail mechanism, is specific to primitive geometric representations, such as polygonal representations.**
- **Continuous Level Of Detail provides a mechanism for specifying an arbitrary number of levels of finer and finer detail, where it is up to the consumer to decide under what conditions the end application should switch from a coarse representation to a more detailed one, or vice versa.**

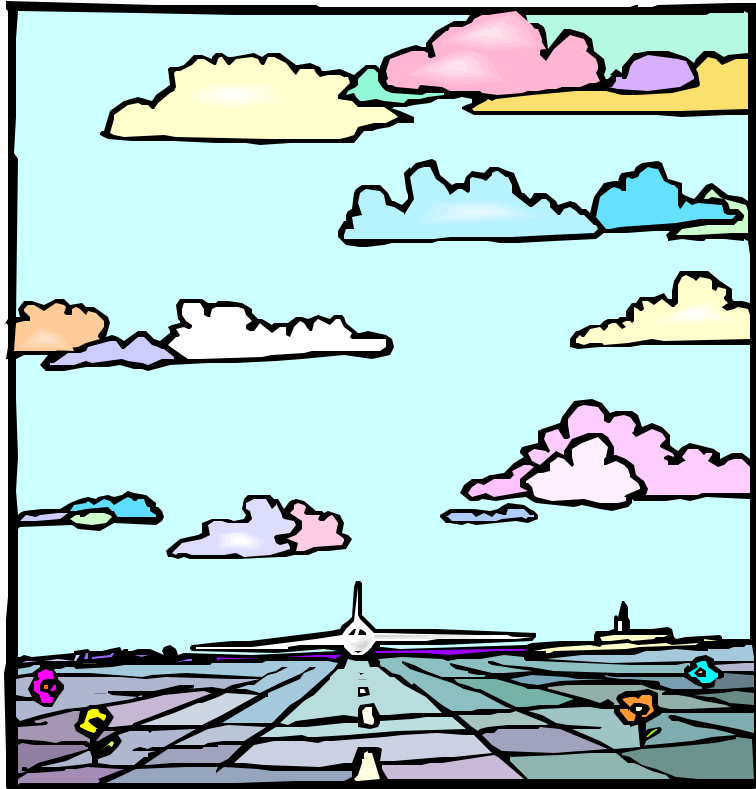
Organizing Principles: Separating Plane & Animation

- **These organizing principles are specific to geometric data.**
- **Some geometric data sets organize geometry by separating planes - that is, planes are specified to partition the geometry into volumes to assist the rendering process.**
- **Some geometric data sets organize geometry to create animation sequences. Such organizations are used to represent environmental events such as explosions.**

Organizing Principle: Time

- A Time Related aggregate organizes data temporally, rather than spatially, where the <Time Constraints Data> on each branch specifies a different time period, such as
 - a <Time Point>
 - a <Time Interval>
 - a <Time Of Day>
 - a <Season>
- For example, a polygonal representation of a forest might use different texture maps, foliage density properties, and so on for different seasons of the year.
- See the atmospheric portion of the *Advanced Application of the DRM* tutorial for a detailed discussion of the uses of time organization as it relates to forecast data.

Organizing Principle: State



- A State Related aggregate uses a special type of EA - a state code - to indicate what state is being represented. Some examples are:
 - EAC_GENERAL_DAMAGE_FRACTION
 - EAC_RUNWAY_SURFACE_CONDITION
- The aggregate indicates what state is being represented and which state is active.
- Each branch represents the object in the state specified by its link object.

Example: State

- A <State Related Geometry> representing different states of a runway for **EAC_RUNWAY_SURFACE_CONDITION** might organize different representations of the runway for
 - **CLEAR** - The runway surface has no weather-related obstruction.
 - **SNOW** - Snow is present on the runway surface.
 - **TOTAL_ICE** - The runway is covered by ice.
- Many state-related aggregates use the <State Control Link> mechanism to express how state changes are determined.

Example: State and <State Control Link>

- Briefly, any <Control Link>, including a <State Control Link>, controls some value in a target object - in this example, a <State Related Geometry>'s active_state value.
- The control itself is specified by some <Expression> provided by the <Control Link>.
- In this example, the <Expression> might be a <Variable> representing the precipitation rate, for which values could be supplied dynamically by the end user, if desired.

Summary of Organization

We've now covered

- **feature, geometric, and tabular representations, including topology**
- **how to organize such representations**
 - **the fundamental organizers**
 - **the more sophisticated organizing principles**

<Transmittal Root> Organization Topics

- **<Environment Root>**
 - We'll discuss the key classes needed for <Environment Root> to represent an environment, illustrated by examples.
 - After covering the lowest-level organizing semantics of such representations and the primitives they organize, we'll discuss the remaining organizing principles.
- **<Library>**
 - The functionality of some key classes related to the <Library> mechanism, with examples.
- **Metadata**
 - We'll discuss the key classes needed to support the metadata that <Transmittal Root> instances must provide, and how consumers can use some of that information.

When Data Exists in the Environment, But...

- **Suppose an object exists in the domain being represented that isn't a fixed part of the environment.**
 - A vehicle
 - A person
 - An animal
- **Suppose a representation exists of some 'generic' thing for which lots of copies will be made.**
 - A house
 - A tree
 - A streetlight
- **How can we store a common representation of something and reference it where, when, and as often as it is needed?**

<Library> and Its Subclasses

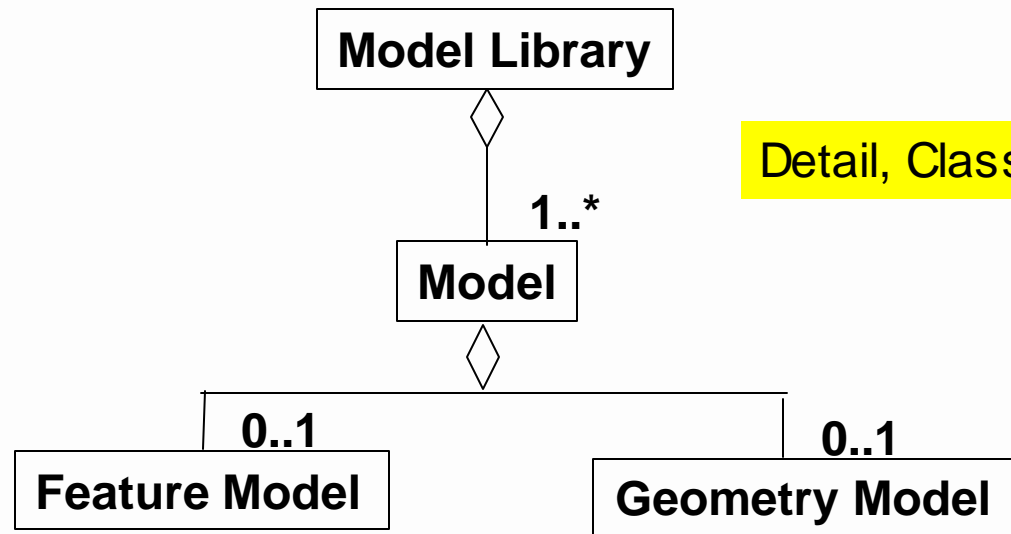
<Library> is a DRM class that abstracts such a concept. Each of its subclasses organizes instances of a particular class.

- <Colour Table Library>**
- <Data Table Library>**
- <Image Library>**
- <Model Library>**
- <Property Set Table Library>**
- <Sound Library>**
- <Symbol Library>**

Referencing the Contents of <Library> Organizations

- For each subclass of <Library>, the DRM has one or more separate classes providing a mechanism for referring to the contents of that kind of <Library>.
 - <Colour Index>
 - <Property Set Index>
 - <Property Table Reference>
 - <Geometry Model Instance>, <Feature Model Instance>
 - <Image Mapping Function>
 - <Sound Instance>
 - <Label>
- Where the contents of tables are being referred to, the corresponding mechanisms provide a way to refer to the contents of the tables, not just the tables themselves. Otherwise, the mechanisms are similar.

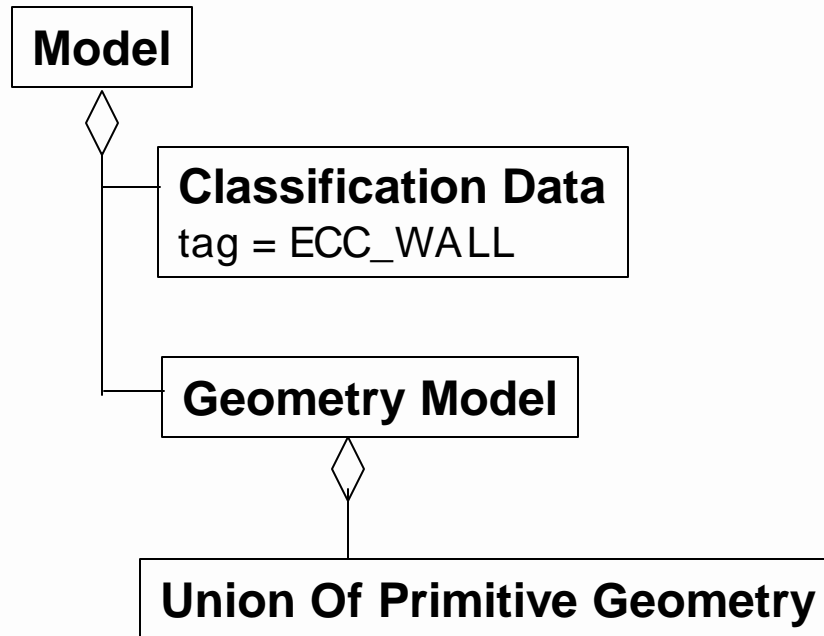
The <Model Library> and <Model> Classes



Detail, Class Diagram Sheet 2

- A <Model> specifies its own spatial frame of reference, which is usually LSR - a 'Cartesian' reference frame.
- A <Model> object may specify a <Feature Model>, a <Geometry Model>, or both.
- The following examples will discuss <Model> solely in terms of <Geometry Model>, because the model instancing mechanism for <Feature Model> is the same.

<Model> - Building Example



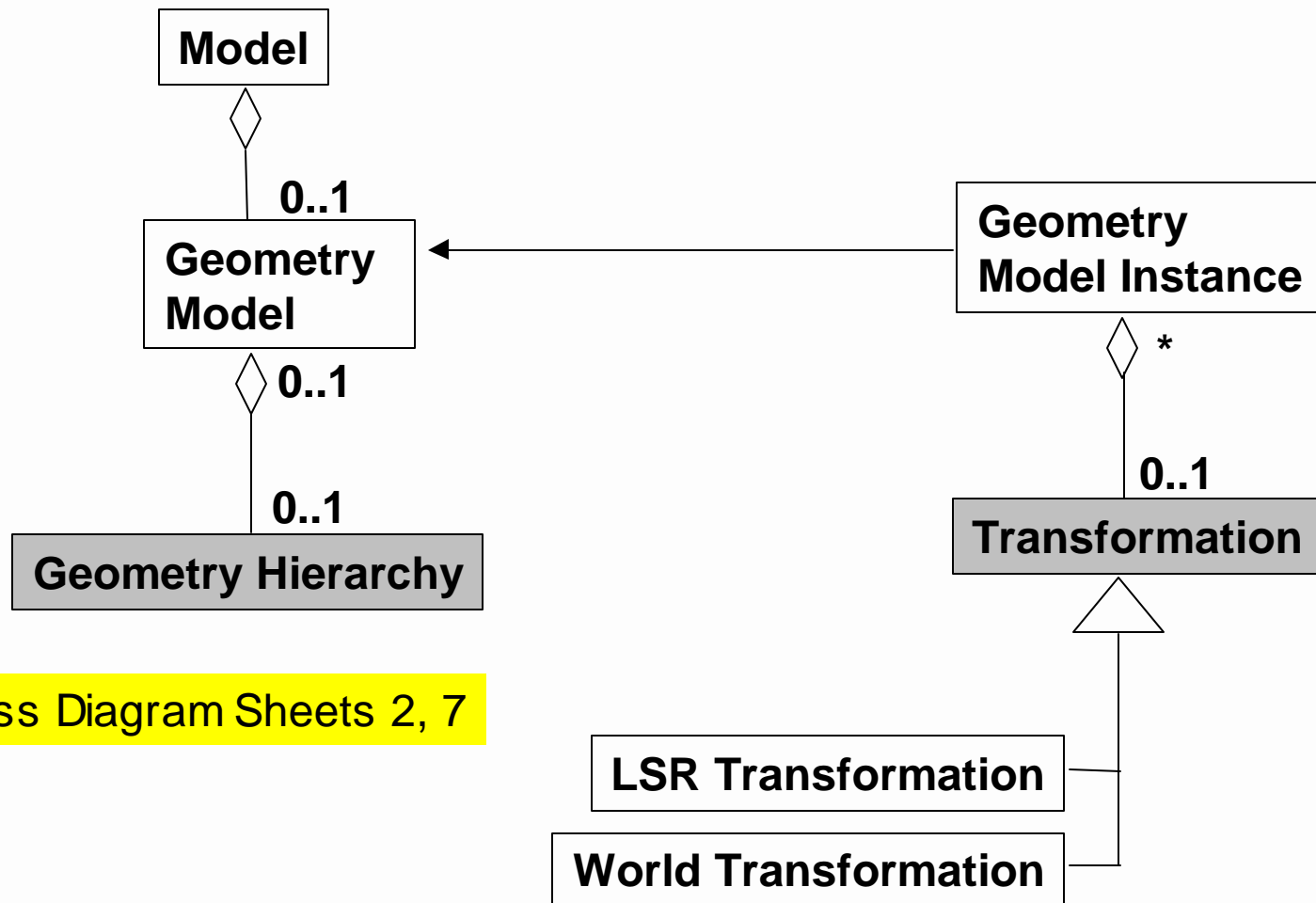
- This <Model> defines the spatial reference frame, usually 3D LSR, in which the wall's geometric representation is defined.
- The geometry of the wall's representation is in the component tree of the <Geometry Model>.

Instance Diagram

Distinguishing Component Parts of a <Model>

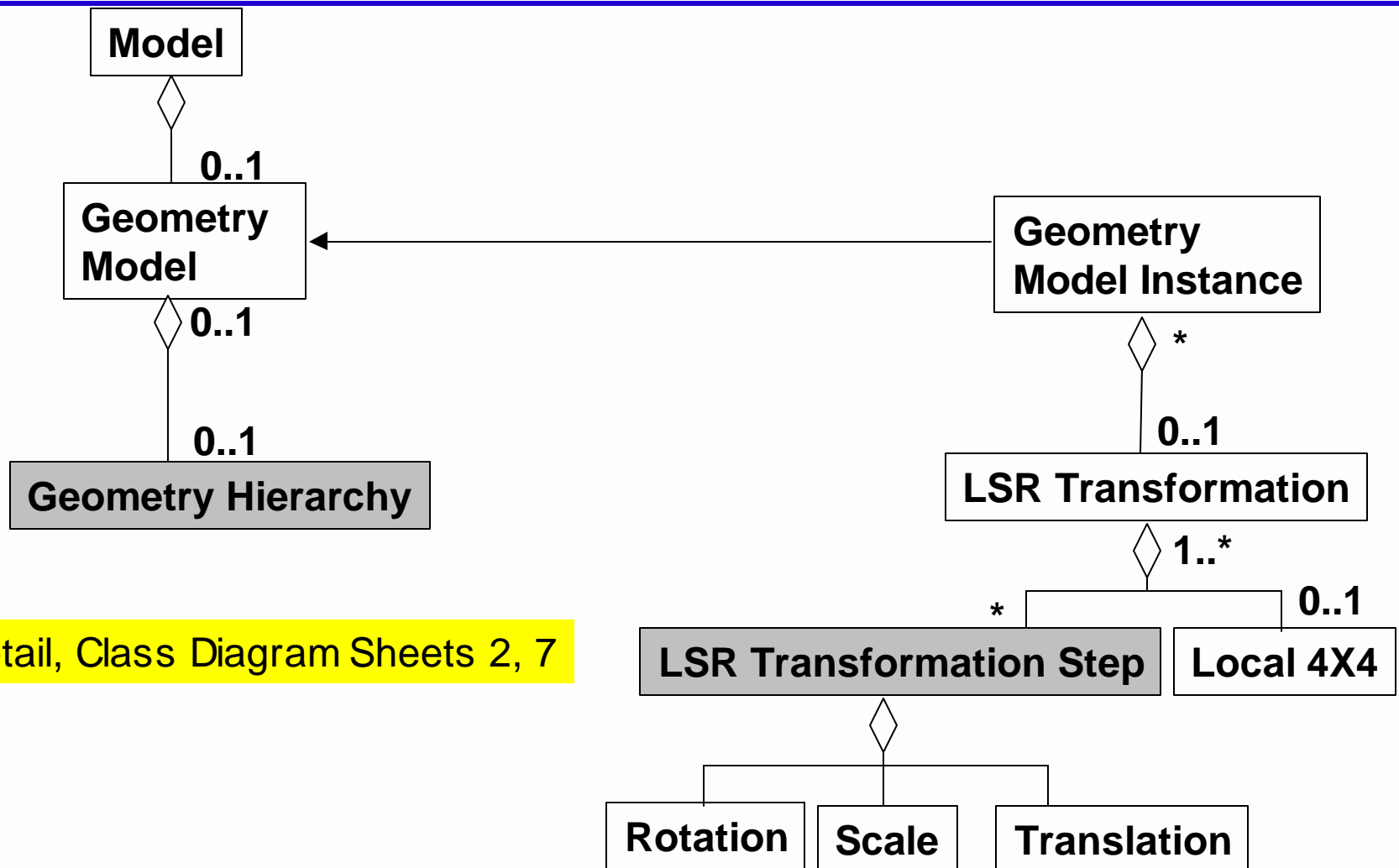
- **Decomposing a <Model> into component models allows the data provider to**
 - reuse component parts
 - distinguish moving parts
- **Component vs. root model**
 - component model - can be instanced only in the scope of another <Model>
 - root model - can be instanced only in the scope of an <Environment Root> instance
 - both - can be instanced in the scope of either a <Model> or an <Environment Root>
- **One catch: Each <Model> has its very own spatial reference frame. How do we resolve this?**

<Geometry Model Instance> and <Transformation>



Detail, Class Diagram Sheets 2, 7

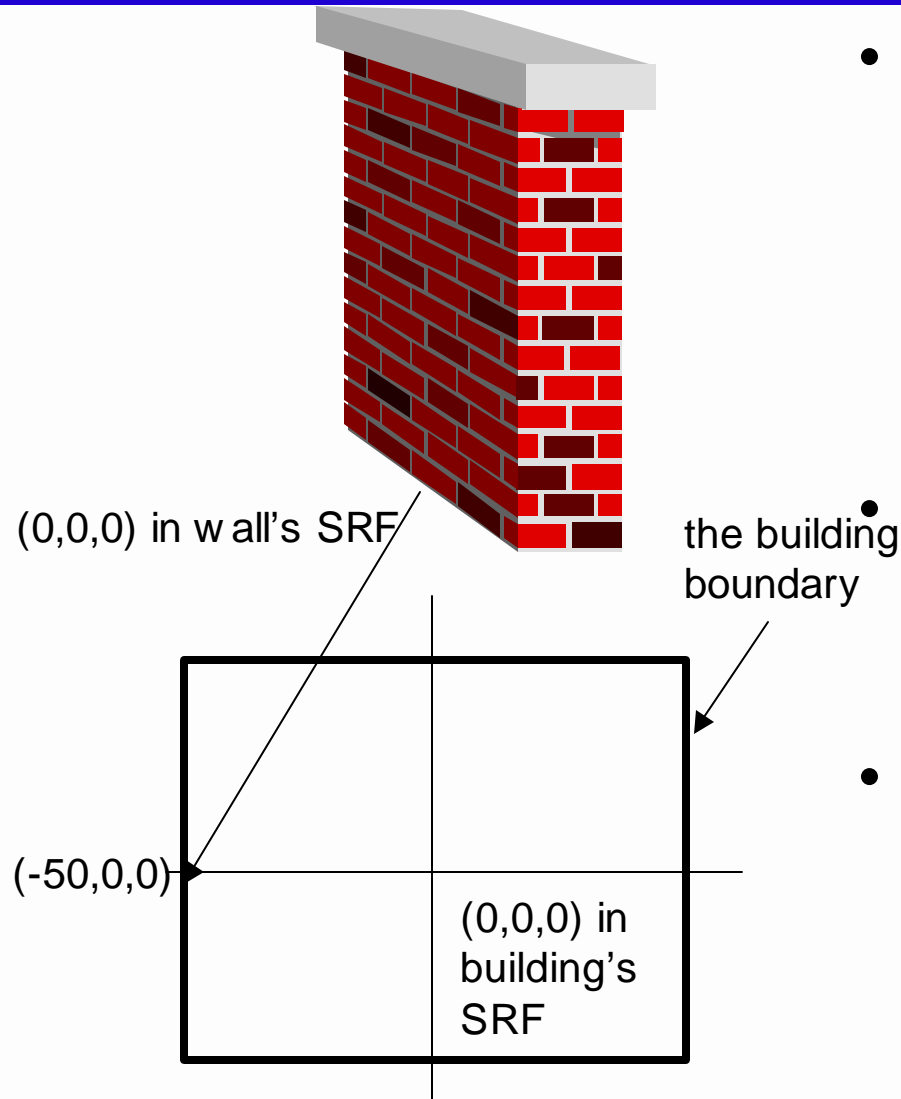
<Geometry Model Instance> and <LSR Transformation>



Detail, Class Diagram Sheets 2, 7

<LSR Transformation>

Example for Model Instancing



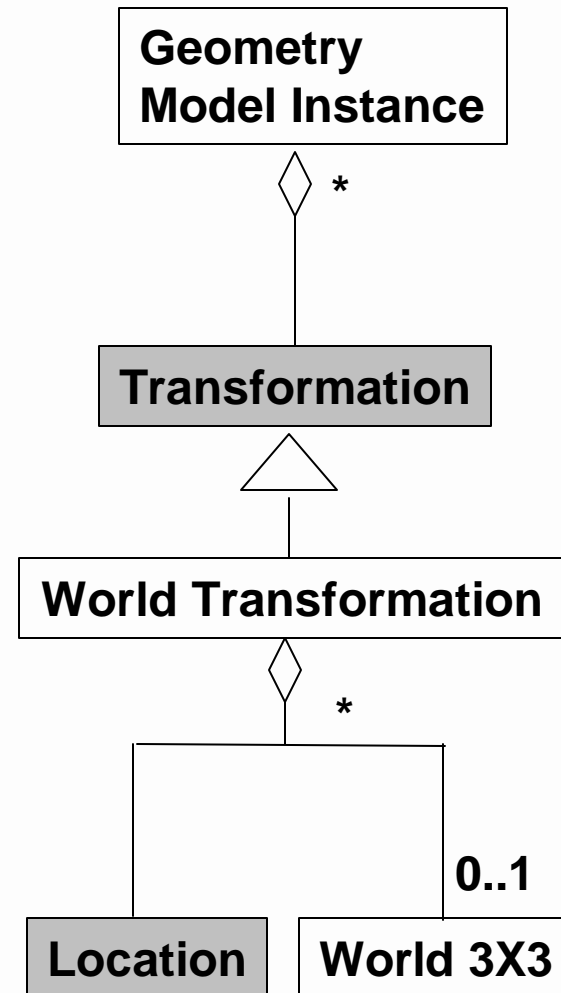
- <LSR Transformation> is used only when
 - both source and destination are LSR
 - source's axes are to be transformed as necessary to align with those of the destination
- The source will then be translated, rotated, and/or scaled as specified to transform it into the target spatial reference frame.
- The <LSR Transformation> may be specified by a series of steps, or by a 4x4 matrix equivalent to such a series of steps.

<Model>s Used in the World

- **In practice, <Model>s tend to appear in one of 2 flavors**
 - **those that appear only in the <Model Library>, because they represent things that usually won't be chained down to a single physical location by the end user, e.g. vehicles, people**
 - **those that represent relatively stationary things, e.g. trees, bridges, buildings, for which a number of copies are desired in the environment**
- **We will introduce a second <Model> example, more suitable to being instanced within an <Environment Root>**

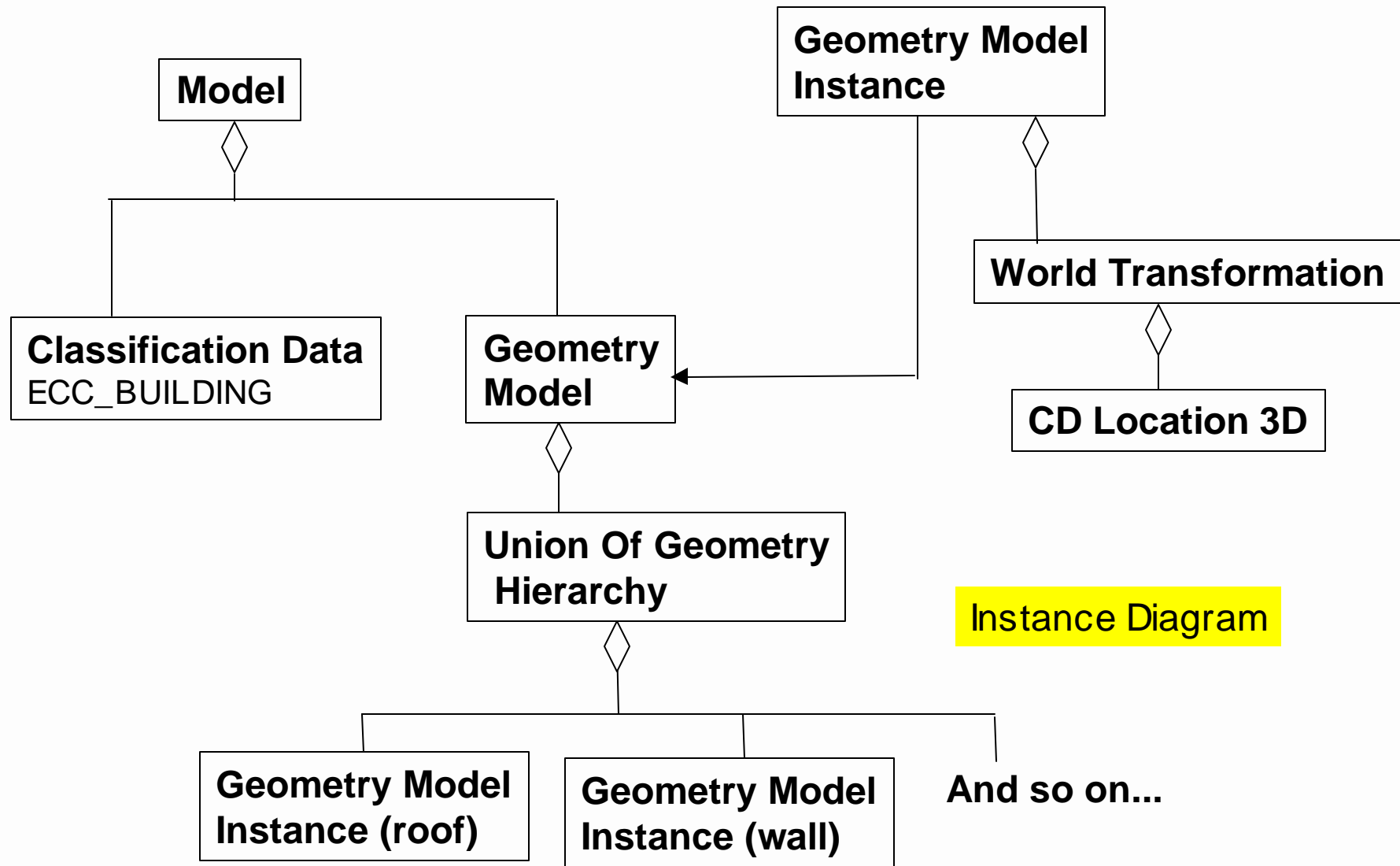
<World Transformation> and Model Instancing

- A <Model> defined in an LSR spatial reference frame has some origin (0, 0) or (0, 0, 0).
- The <World Transformation> specifies the <Location> in the target spatial reference frame where the LSR reference frame's origin will be instanced.
- The optional <World 3X3> matrix specifies any further adjustments to be made.



Detail, Class Diagram Sheet 7

<Geometry Model Instance> in the 'World'



<Transmittal Root> Organization Topics

- **<Environment Root>**
 - We'll discuss the key classes needed for <Environment Root> to represent an environment, illustrated by examples.
 - After covering the lowest-level organizing semantics of such representations and the primitives they organize, we'll discuss the remaining organizing principles.
- **<Library>**
 - The functionality of some key classes related to the <Library> mechanism, with examples.
- **Metadata**
 - We'll discuss the key classes needed to support the metadata that <Transmittal Root> instances must provide, and how consumers can use some of that information.

Metadata in the DRM

- **Metadata in the DRM comes in 2 "flavors" - information about transmittal content designed purely for human users, and summary information that can also be used by consuming software.**
- **The first variety corresponds to the ISO metadata standard.**
 - **<Access>**
 - **<Responsible Party>**
- **The second variety consists of the "summary" classes, which can be used to summarize the contents of a transmittal in a form that can be understood by a consuming application as well as by human readers.**
 - **<Transmittal Summary>**
 - **<Environmental Domain Summary>**

CSGDM-Compliant Metadata

- **Every <Transmittal Root> instance is required to specify**
 - **<Responsible Party>**
 - **<Access>**
 - **<Citation>**
 - **<Description>**
 - **<Keywords>**
 - **<Data Quality>**
 - **a <Base Time Data>, such as the creation time of the transmittal**
- **More information may be specified, both for the <Transmittal Root> and at other levels of a transmittal, but this is the minimum required information.**

Summary Metadata

- Every <Transmittal Root> instance is required to specify a <Transmittal Summary> component.
- <Transmittal Summary> provides a brief overview of the contents of the transmittal in a manner that can be understood in software, including
 - what kinds of libraries are present
 - whether feature data is present, and where
 - whether geometry data is present, and where
- Further summary information may be provided, but this is the minimum required.

Where To Go From Here

- Review the DRM dictionary and HTML pages, along with the DRM class diagrams.
- Develop instance diagrams using the basic techniques you have learned here and based on specific application topics of interest to you.
- Attend 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.
- Attend the “*Advanced Use of the SEDRIS SDK*” tutorial, if you are interested in working with software to use, store, or manipulate DRM data.
- Attend the “*How to Produce and Consume Transmittals*” tutorial, if you are interested in producing or consuming DRM data.