

How To Produce & Consume Transmittals

<http://www.sedris.org>

SEDRISTM Technology Conference
Lake Buena Vista, FL
9 January 2004

Jesse Campos
SAIC
jesse.j.campos@saic.com



Introduction

- **DESCRIPTION** – This tutorial is conducted in two consecutive parts. Part 1 covers creation and writing of SEDRI transmittals. Part 2 focuses on accessing or extracting data from SEDRI transmittals. Each part is conducted independent of the other. Common application development techniques and strategies utilized in the production and consumption of SEDRI transmittals are covered. The steps in development of mapping documents and the effective use of the SEDRI API are discussed. A number of examples based on actual use cases are reviewed.
- **WHO SHOULD ATTEND** – Software engineers who intend to develop tools, utilities, or conversion applications to operate on SEDRI transmittals.
- **PREREQUISITE** – Prior attendance at the "Fundamentally SEDRI: The Technology Components" tutorial, the "Fundamentals of the DRM" tutorial, and the "Developing Effective Applications with the SEDRI API" tutorial is recommended.
- **WHAT TO EXPECT** – At completion, the attendee should have a working understanding of the various how-to techniques for use of the SEDRI API in the creation or extraction of transmittals.



Prerequisite

- To get the most from this tutorial, we assume you know the following information as a prerequisite to this session:
 - Solid understanding of the SEDRIIS technology components.
 - Have attended at least one of the *Fundamental of the DRM* and *Fundamentals for Accessing Transmittals* tutorials.
 - Solid understanding of software development principles.



Putting It All Together

- **Use all of the SEDRIIS Technology Components**
 - Data Representation Model (DRM)
 - Spatial Reference Model (SRM)
 - Environmental Data Coding Specification (EDCS)
 - Interface Specifications (API)
 - SEDRIIS Transmittal Format (STF)
- **Combine with**
 - SEDRIIS tools & applications
 - SEDRIIS documentation

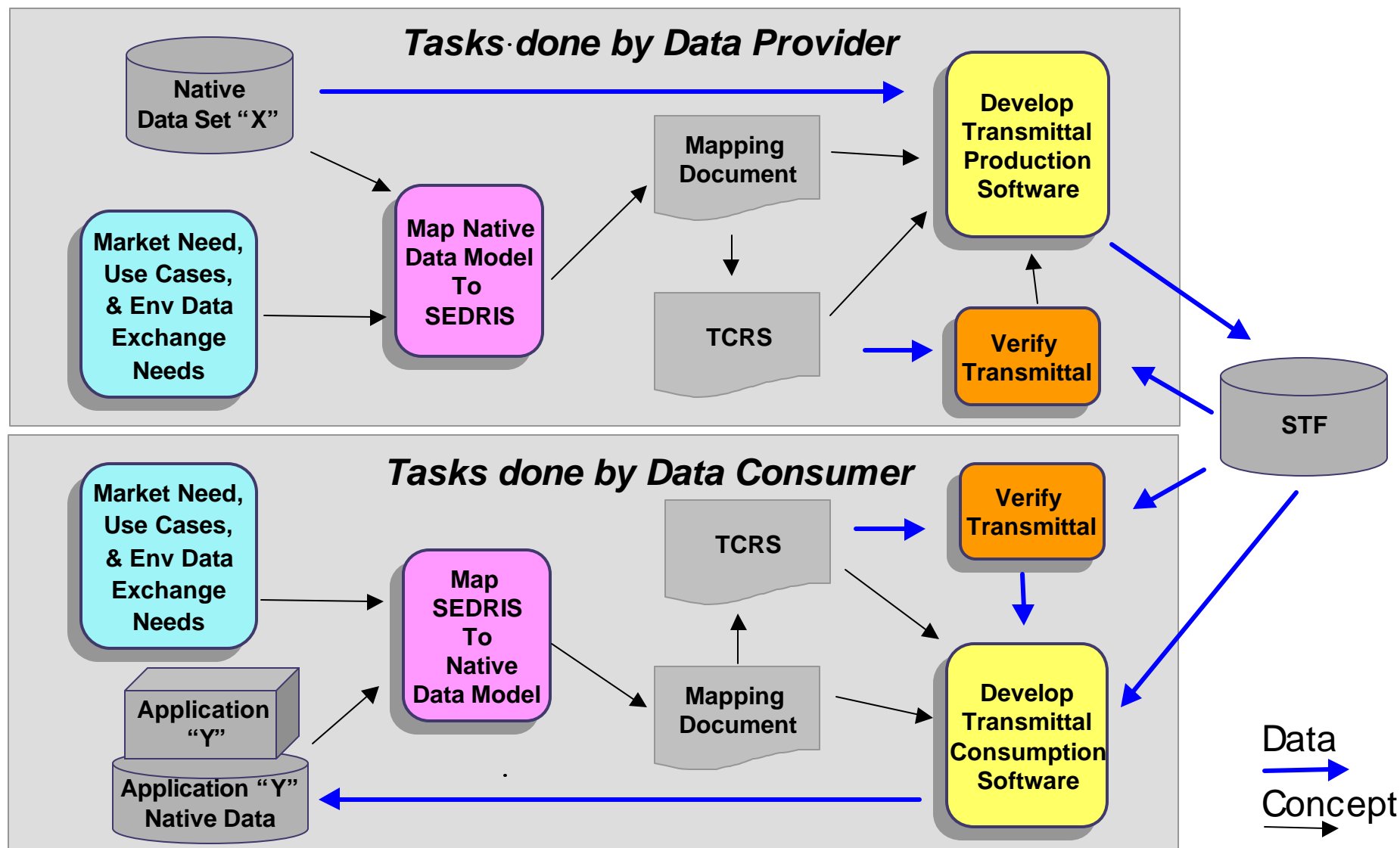


Implementation Versus Design

- **Implementation**
 - *Fundamentals of the DRM* Tutorial
 - *Fundamentals for Accessing Transmittals* Tutorial
 - Focus: Mechanics
- **Design**
 - Techniques
 - Common strategies
 - Products required
 - Focus: the application of knowledge gained; i.e., wisdom



Production and Consumption Process



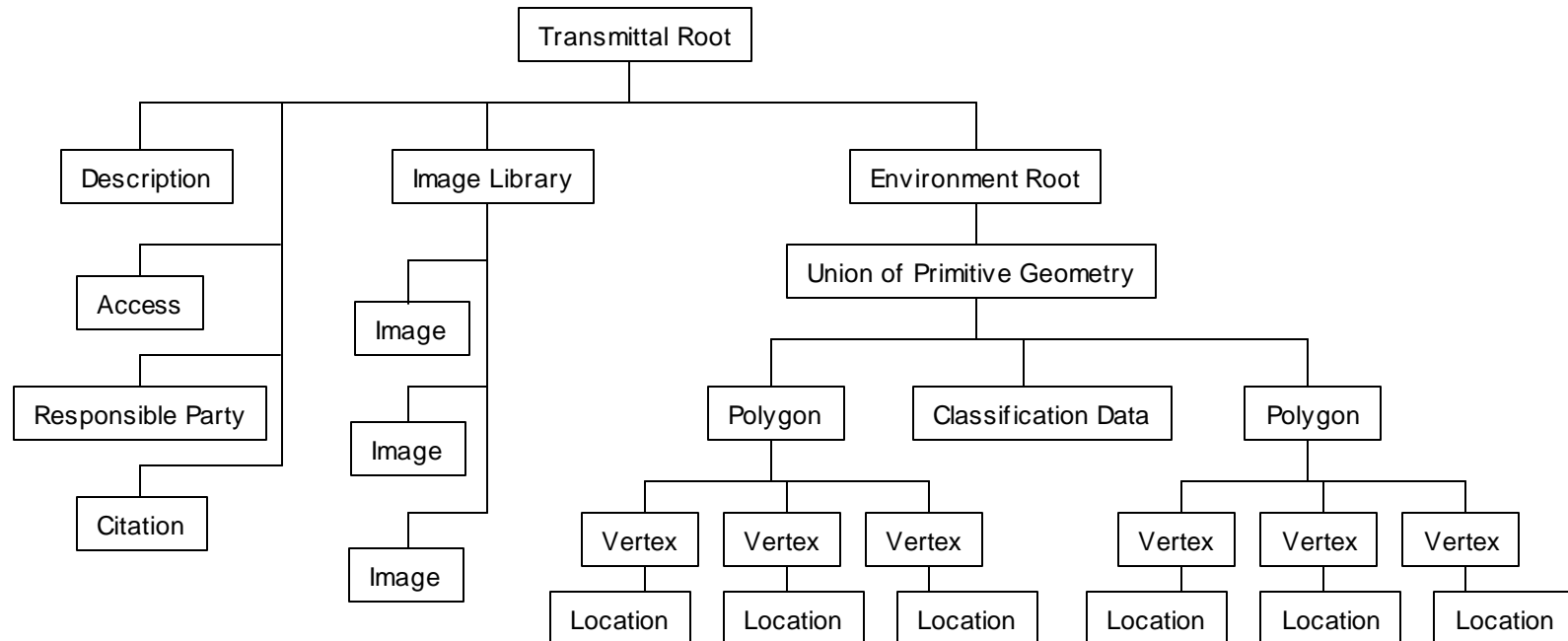


Agenda: Producing

- **SEDRIIS Transmittals**
- **Transmittal Creation Process**
 - **Native analysis**
 - **The mapping document**
 - **Feature versus geometry representation**
 - **Object sharing**
 - **Utilizing component inheritance.**
 - **Transmittal metadata**
 - **<Hierarchy Summary Item>**
 - **Develop Validation Criteria**
 - **Using the SEDRIIS Write API**
 - **Validation and testing utilities**



SEDRI Transmittals

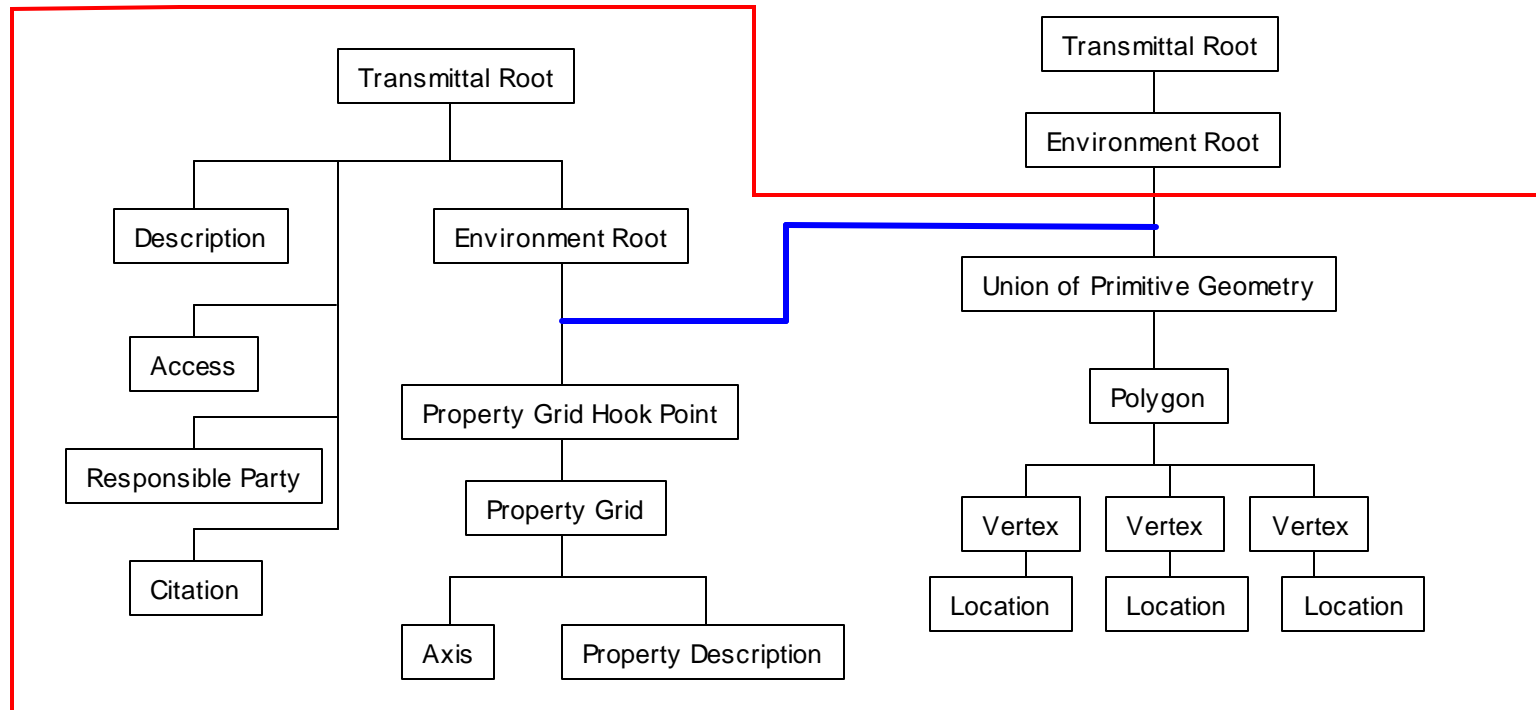


Sample Transmittal

- A realization of environmental data a collection of DRM-compliant instances accessible through the SEDRI API
- Transmittals must satisfy the DRM and as such are complete unto themselves



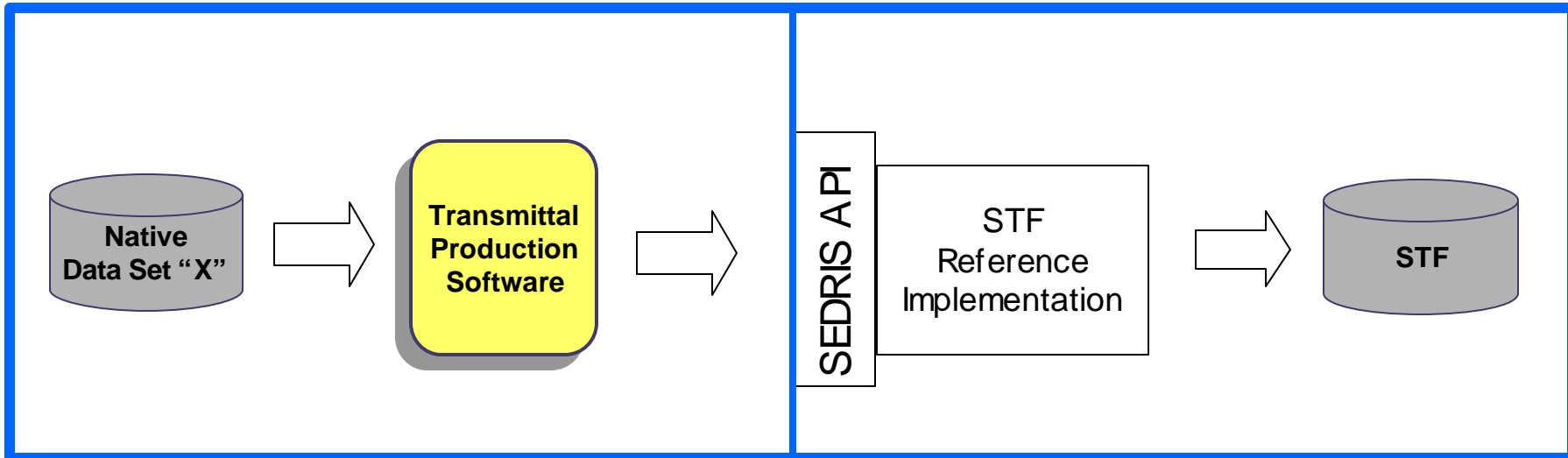
SEDRIIS Transmittals [2 of 5]



- A Transmittal may reference data in another Transmittal
- This capability is termed Inter Transmittal Referencing (ITR)
- i.e., there is an ITR reference between the above two transmittals



SEDRIIS Transmittals [3 of 5]

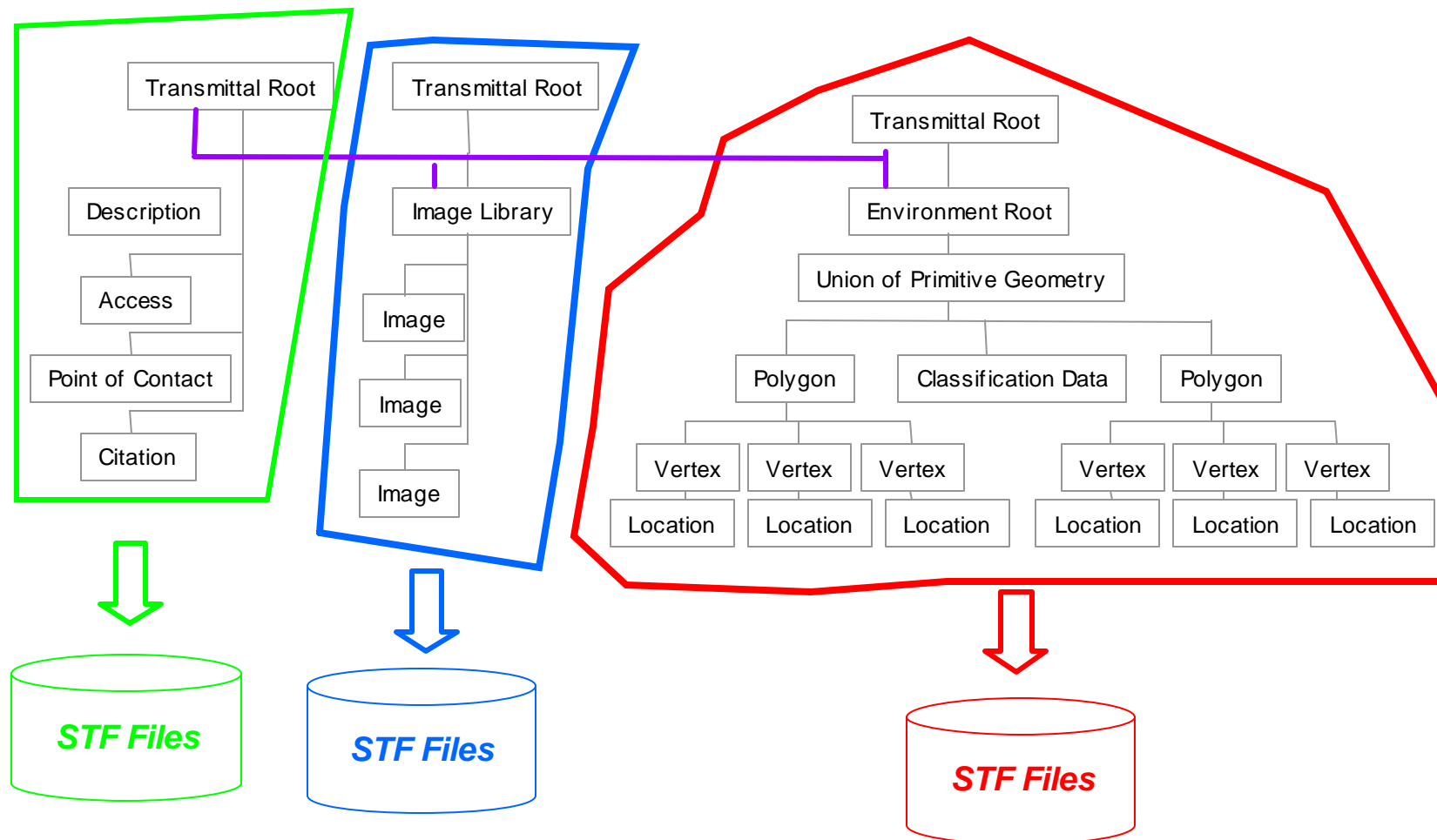


- **Using the SEDRIIS SDK**
 - Transmittals are stored in STF
 - Consists of a set of STF files
 - One root file used for opening
 - Variable number of object files based on the count of objects in the transmittal
 - Variable number of <Image> and <Data Table> data files



SEDRIS Transmittals [4 of 5]

- ITR in the SDK



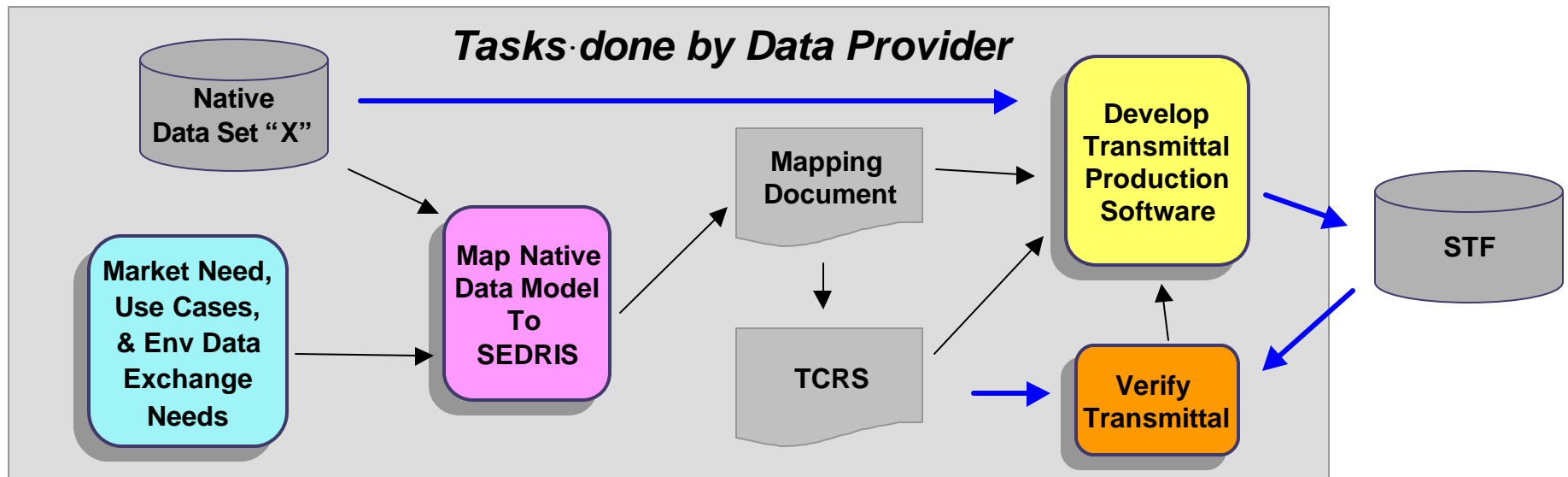


SEDRIIS Transmittals [5 of 5]

- **What is in a (transmittal) name**
 - **Produced Name –**
 - Stored in the transmittal in the <Transmittal Root> field value
 - SE_STRING
 - **File Name**
 - Provided through the API when opening in Create mode
 - Assigned to the STF root file by the STF implementation
 - **URN Name**
 - Uniform Resource Name
 - Resolves to a Universal Resource Locator (URL)
 - Used to create ITR references
 - Example: urn:x-sedris:saic:bellevue:1
 - Assigned through the API, stored in STF root file



The Process of Creating a Transmittal



Step 1: Native Requirements & Data Analysis: Define use or application plus data exchange requirements

Step 2: Develop Mapping Document: Use DRM, EDCS and SRM

Step 3: Develop Validation Criteria -- TCRS

Step 4: Develop Production software: Add in API and STF

Step 5: Validate Transmittal: Add in tools and applications

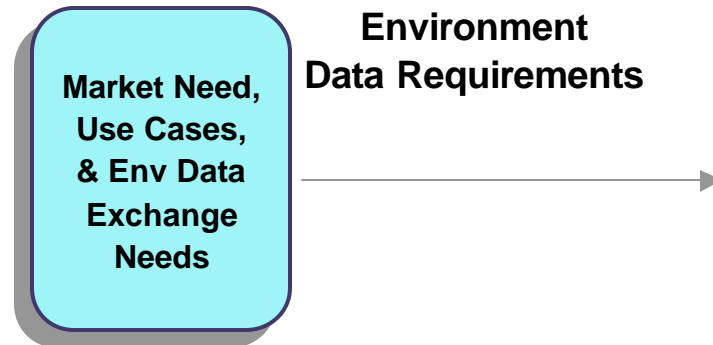


Step 1: Native Analysis

Goal: Capture all known and potential uses for native data.

Analyze:

- use cases
- application requirements
- data content and utility
- data customer needs and applications
- data deficiencies (e.g. metadata, data augmentation, ...)



Determine best way to maximize market potential for data being produced in STF

Use the analysis to focus your mapping efforts in Step 2



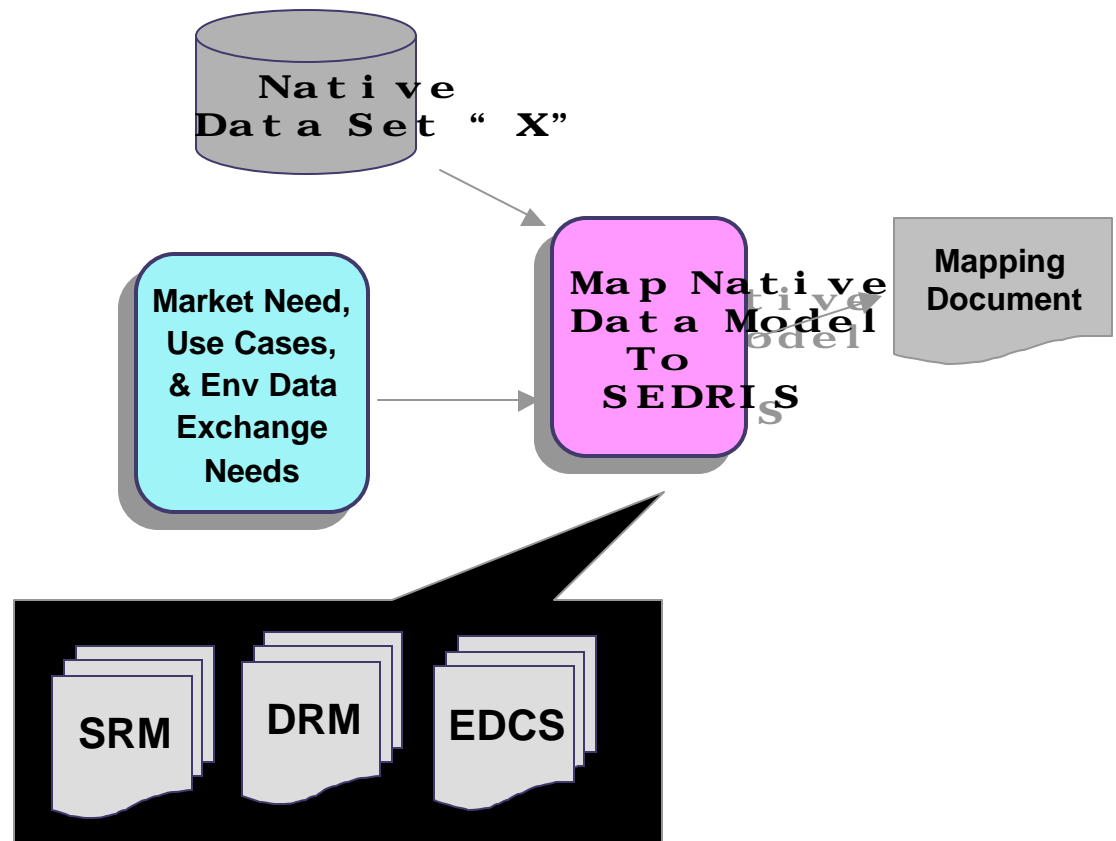
Step 2: Mapping Document

Purpose

- Document the translation from a native data model to DRM
- Find any problems the DRM, EDCS, or SRM does not handle with your production application

Goals:

- Map native data organization to the DRM
- Use correct EDCS entries
- Document mapping
- Provide design criteria for software development
- Provide validation criteria for STF content





Step 2: Mapping Document [2 of 3]

- **First: List**
 - primitive data elements within your native data
- **Second: Categorize**
 - Primitive data (locations, polygons, lights, etc.)
 - Organizing elements (a bag of, a tile of, etc.)
 - Descriptions (attributes, classifications etc.)
 - Implementation artifacts (text strings, run time values)
- **Third: Learn**
 - Primitive data classes
 - <Point Feature>, <Linear Feature>, and <Areal Feature>,
 - <Point>, <Polygon>, <Light Source>, <Image>, <Sound>,
 - <Property Grid>, <Property Table>, etc.



Step 2: Mapping Document [3 of 3]

- **Third: Learn**
 - Descriptive classes
 - <Location>, <Color>, <Classification>, <Transformation>, <Properties>
 - Organizing Containers classes
 - Hierarchies, Libraries
- **Fourth: Map**
 - Document the mapping in the document
 - Document the EDCS codes to be used
- **Fifth: Request**
 - DRM
 - SRM
 - EDCS
 - Classifications
 - Attributes



Mapping Document: Features versus Geometry

- **What type of data do you have?**
 - Visual
 - Measured
 - Abstract
 - SAF
- **Both available**
 - Provide both representations
 - Provide associations between them
 - Allows consumers to correlate the data



Mapping Document : Data Mapping

- **Consumer oriented *versus* “write only”**
 - Map intrinsically environmental parts of your data
 - “Value added” data
 - Discard data implementation artifacts
- **A mapping test**
 - Self contained DRM mapping, Self describing data?
 - Objective choices of classification and attribute codes
 - Clear organizational objectives
 - Mapping Document
 - Sufficient description of discarded artifacts to allow the rebuilding of native data set?



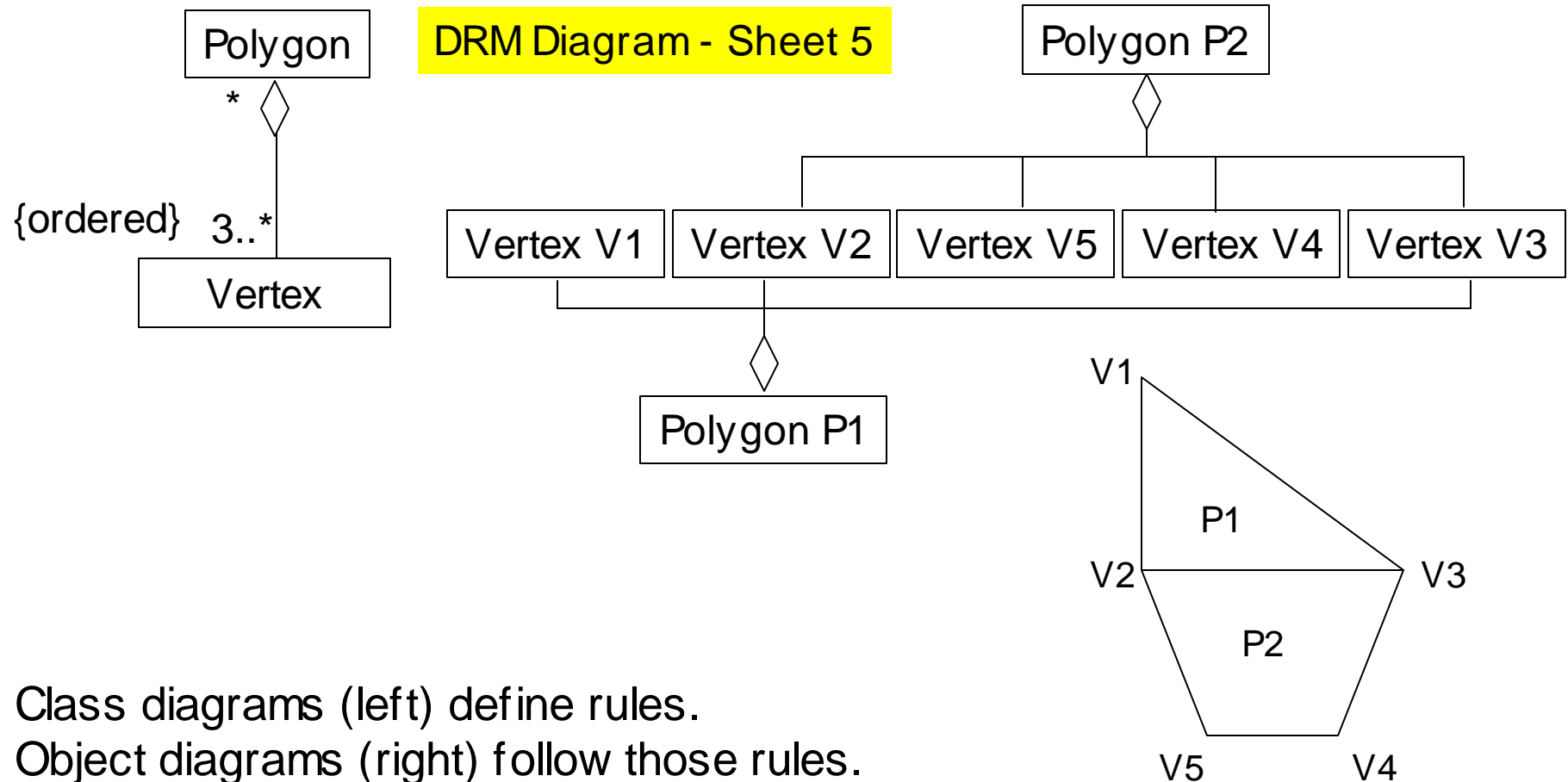
Mapping Document: Object Sharing

- **Saves time and space**
- **Sharing through Aggregation**
 - Re-use the same instance of an object
 - A relationship between objects is a fraction of the cost of new objects
- **DRM constructs for explicit sharing**
 - **<Library>**
 - <Images>, <Models>, <Data Tables>, etc.
 - **<Property Set>**
 - A table of classification, property, rendering & meta data objects
 - **<Colour Table>**
 - A table of primitive colors



Sharing Components: DRM Example

DRM Diagram - Sheet 5



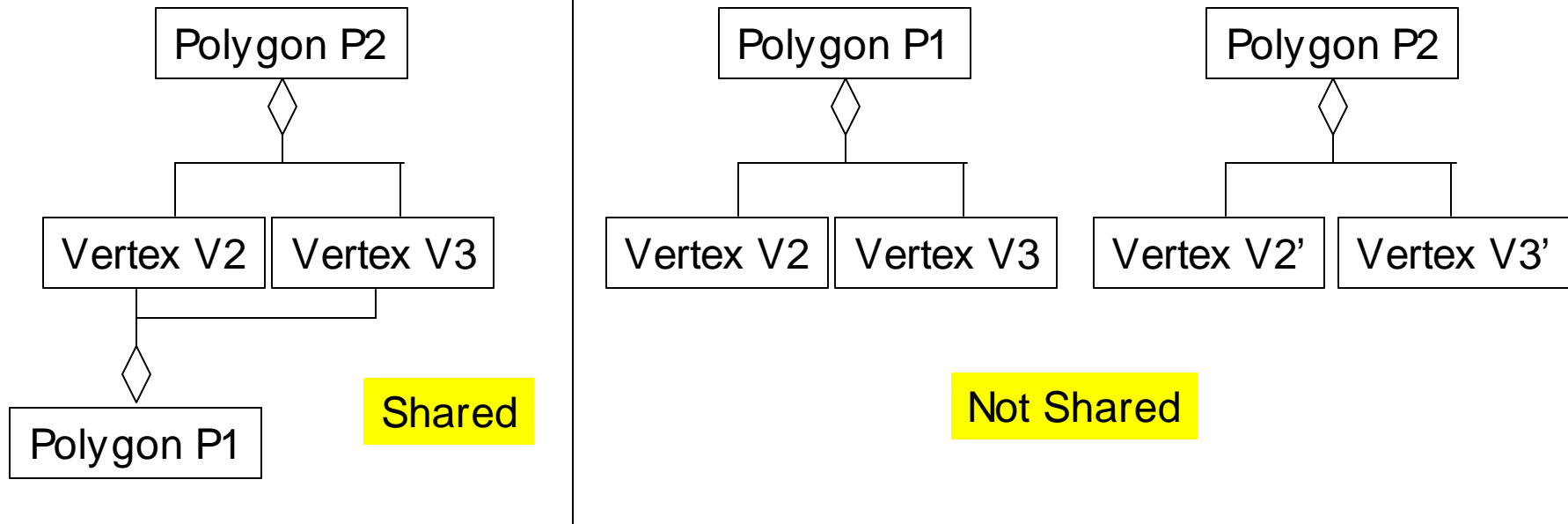
Class diagrams (left) define rules.

Object diagrams (right) follow those rules.

<Vertices> can be shared by <Polygons>, because
a <Vertex> can be a component of 0 or more <Polygons>.



Why Share?



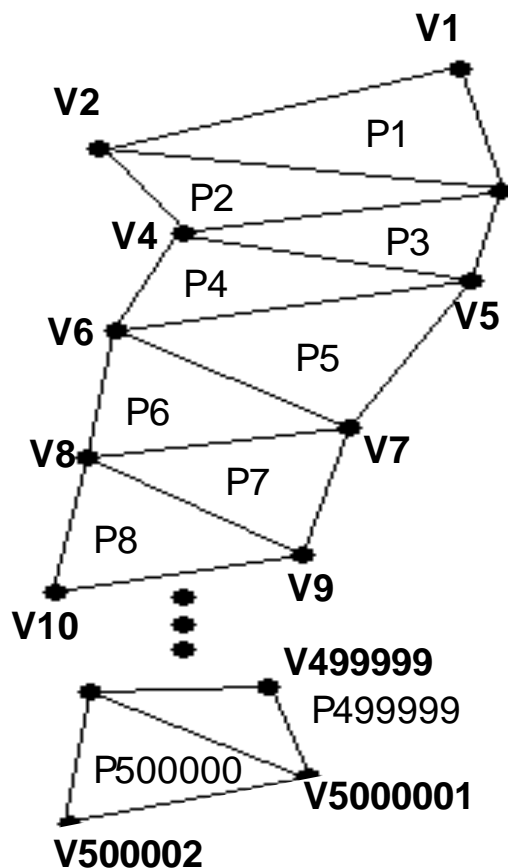
Sharing reduces the number of objects in a transmittal.

In this example, the <Polygons> have a total of 5 <Vertices> if they share their common <Vertices>, but 7 if they don't. This savings may seem trivial, but when you bring this into the realm of large transmittals ...

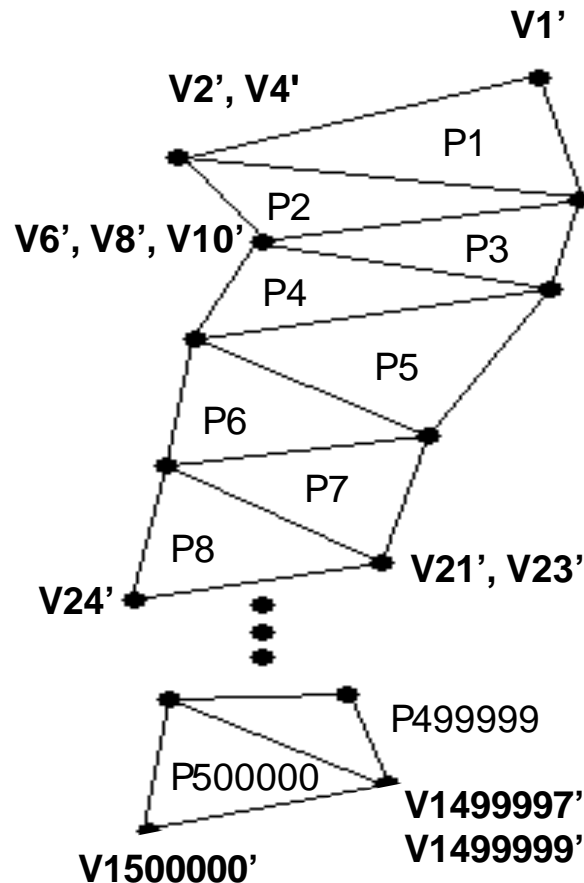


Why Share?

Triangle Strip as an Example of Sharing



Shared



Not Shared

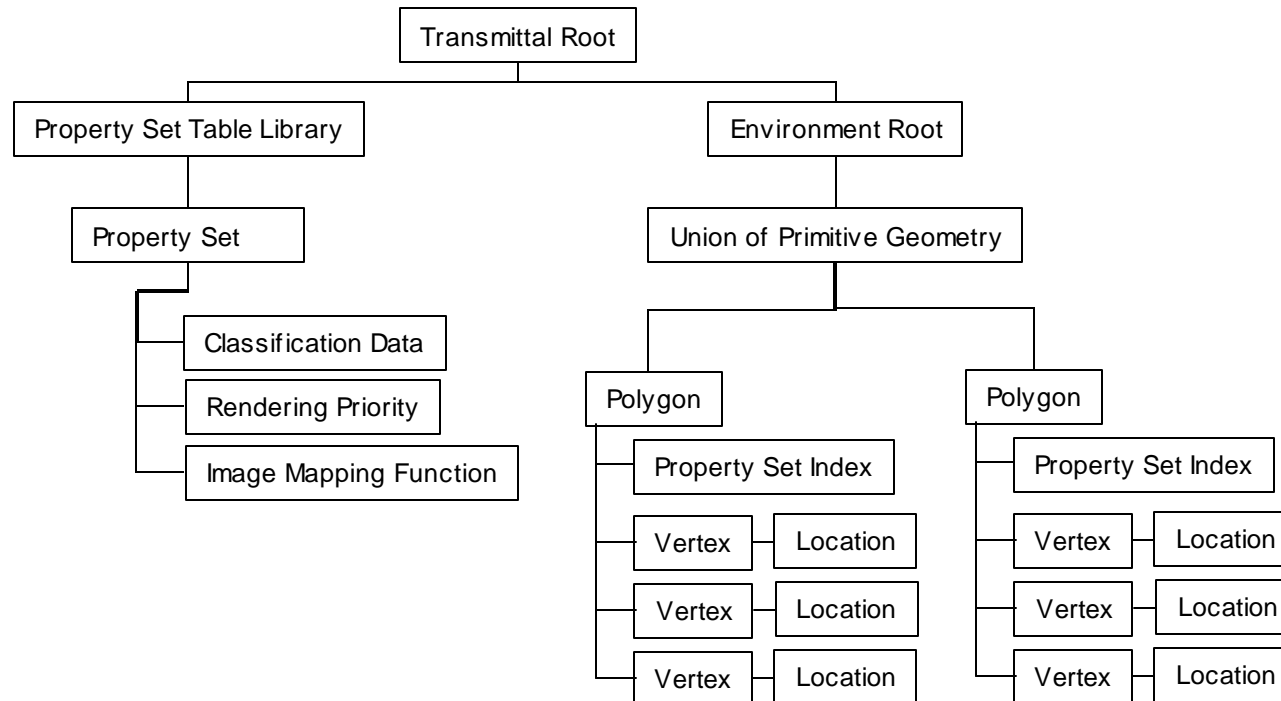
In a triangle strip of 500000 <Polygons>, there are **1.5 million** <Vertices>, if none of the <Vertex> instances are shared.

In a triangle strip such as this, only 2 <Vertex> objects could not be shared, so sharing gives us

$500000 \text{ <Polygons>} * 1 \text{ <Vertex> per <Polygon>} + 2 \text{ <Vertices>} =$
500002 <Vertices>
with a savings of 66%.



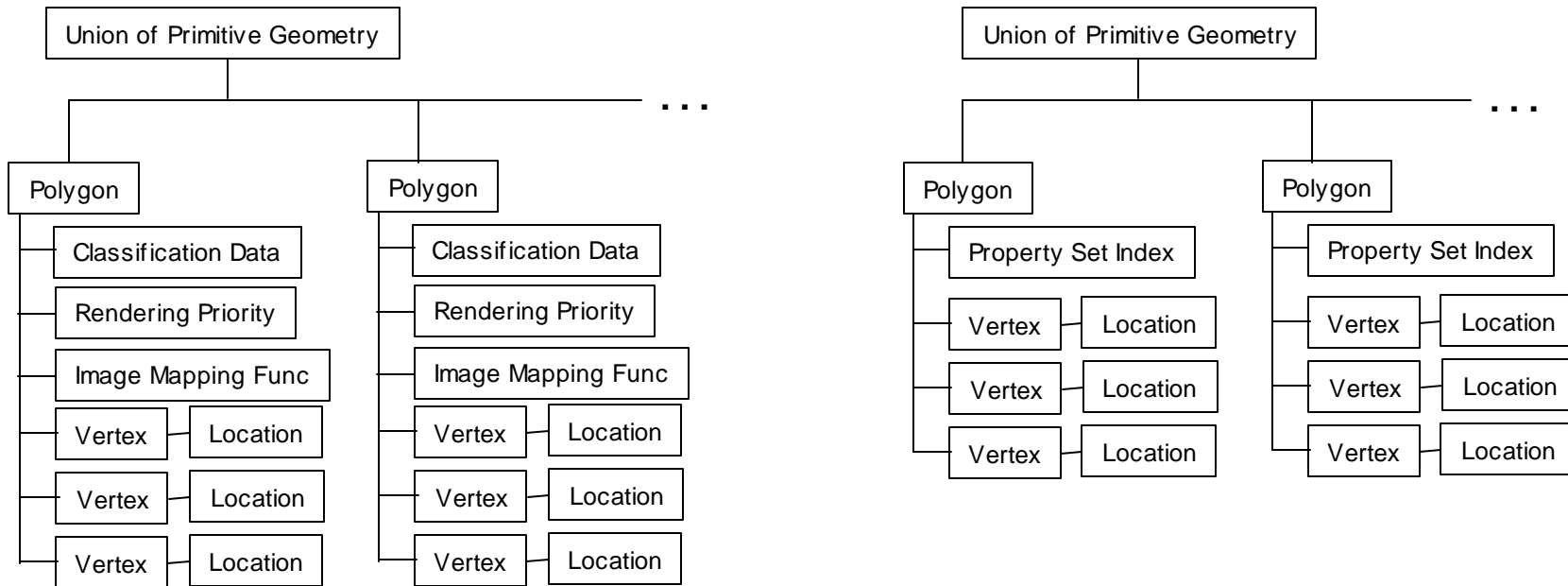
Mapping Document: Sharing with <Property Sets>



- **Group a set of attributes under the <Property Set Table Library>**
- **Use for <Geometry> and/or <Feature>**
- **Reference with an <Property Set Index>**



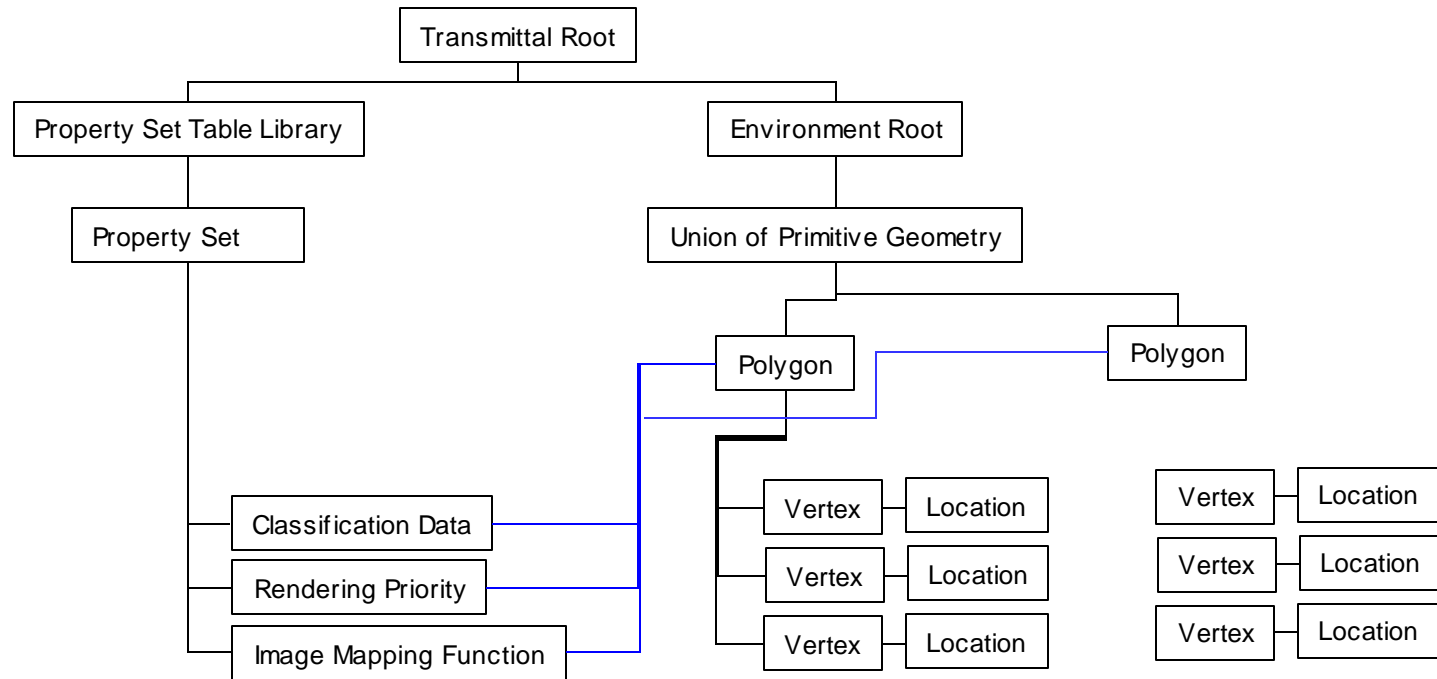
Mapping Document: Sharing with <Property Sets> [2 of 3]



- **Two possible representations providing the same data**
 - Left one duplicates objects
 - Right one uses the <Property Set Index>



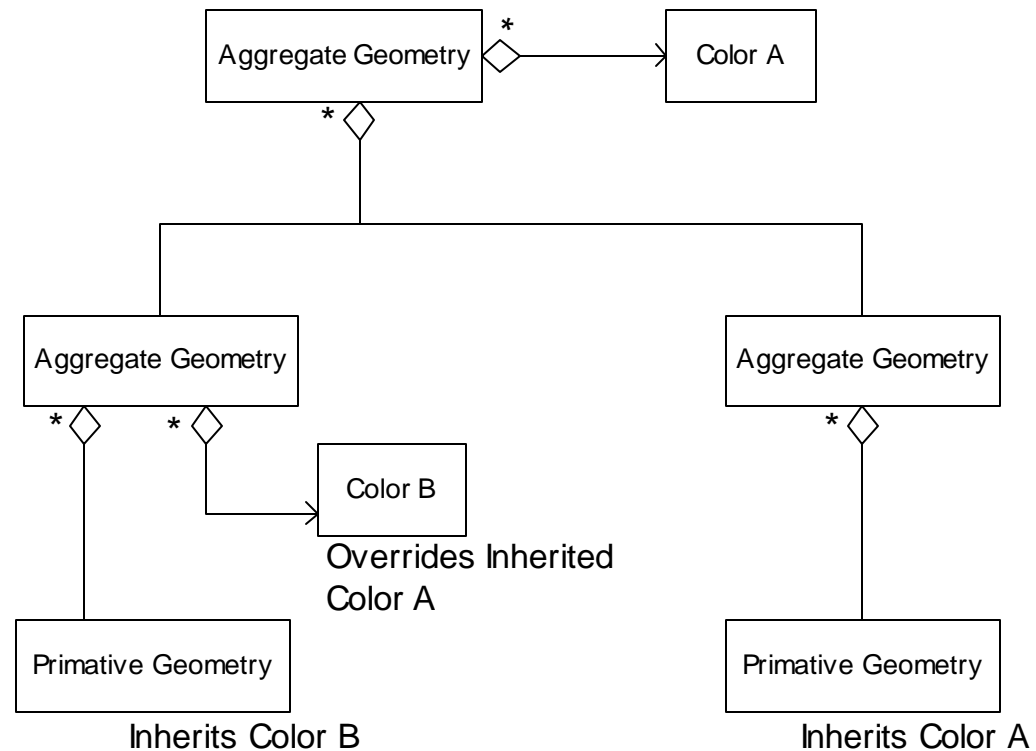
Mapping Document: Sharing with <Property Sets> [3 of 3]



- **Replace the <Property Set Index> with component relationships**
 - Simplifies consumption
 - Space savings equivalent for small sets of properties
 - Recommended method



Mapping Document: Component Inheritance





Mapping Document: Component Inheritance [2 of 3]

- **Components which are inherited from ancestors**
 - **<Access>**
 - **<Classification Data>**
 - **<Data Quality>**
 - **<Light Rendering Properties>**
 - **<Rendering Priority Level>**
 - **<Time Constraints Data>**
 - **<Property Tables>** (overridden per ECC code)
 - **<Property Table References>** (overridden per ECC code)
 - **<Property Values>** (overridden per EAC code)
 - **<Colours>** (overridden per presentation domain)
 - **<Rendering Properties>** (overridden per presentation domain)



Mapping Document: Component Inheritance [3 of 3]

- **Other components affected by inheritance:**
 - **<Property Set Index>**
 - Referenced inheritable components are inherited
 - **<Image Mapping Functions>**
 - At each level overrides as a group
 - **<Location>**
 - Inherited by Reference Vectors
- **Reference:**
 - **SEDRIS Technology Documentation Set**
 - **Part 4: *Technical Reference Set***
 - **Volume 9: *Attribute Inheritance and Context Technical Guide***



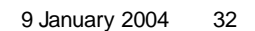
Mapping Document: Transmittal Metadata

- **Attached to <Transmittal Root>**
 - <Access>
 - <Citation>
 - <Description>
 - <Keywords>
 - <Responsible Party>
- **Use data relevant per transmittal**
- **URN Name**
 - Optional through the API
 - Register with SEDRIS organization
 - Uniquely identify the transmittal and the creating organization



Mapping Document: Summary Items

- **<Hierarchy Summary Item>** describes the organization of a Transmittal
 - Can represent the exact structure of a Transmittal
- **<EDCS Summary Item>**
 - Class to define a classification code and a set of attributes
- **<DRM Class Summary Item>**
 - Lists all classes used in a Transmittal
- **<Primitive Summary Item>**
 - Can represent the structure of primitive objects





Step 2: Mapping Document

- **Benefits**
 - Described the conversion from a native format to SEDRIIS
 - Provided a road map for consumers
 - Found any problems the DRM, EDCS, or SRM does not handle
 - Have a clear understanding of what you want your Transmittal to look like
 - Clear documentation to evaluate how well you succeeded
 - Easy place to analyze SEDRIIS specific information
 - The hard part is done



CTDB to STF Mapping Example

- **Converts data designed for Semi-Automated Forces (SAF) application into SEDRIIS**
- **Does not map run time information**
- **CTDB constructs refer to items as features, but DRM requires more decomposition**
 - **Decompose into objects with features**
 - **Decompose into classifications with property values.**



Sample Basic Mappings

CTDB Attribute	EAC
<i>min_x, max_x, min_y, max_y</i>	<Spatial Domain>
<i>num_features</i>	Invalid; not saved
<i>patch_cache_size_map</i>	Not saved
<i>sub_format</i>	Not saved
<i>origin_northing, origin_easting, origin_zone_number, datum,</i>	<Environment Root>
Name	Not saved
version	Not saved
Date	Not saved
num_nodes	Not saved



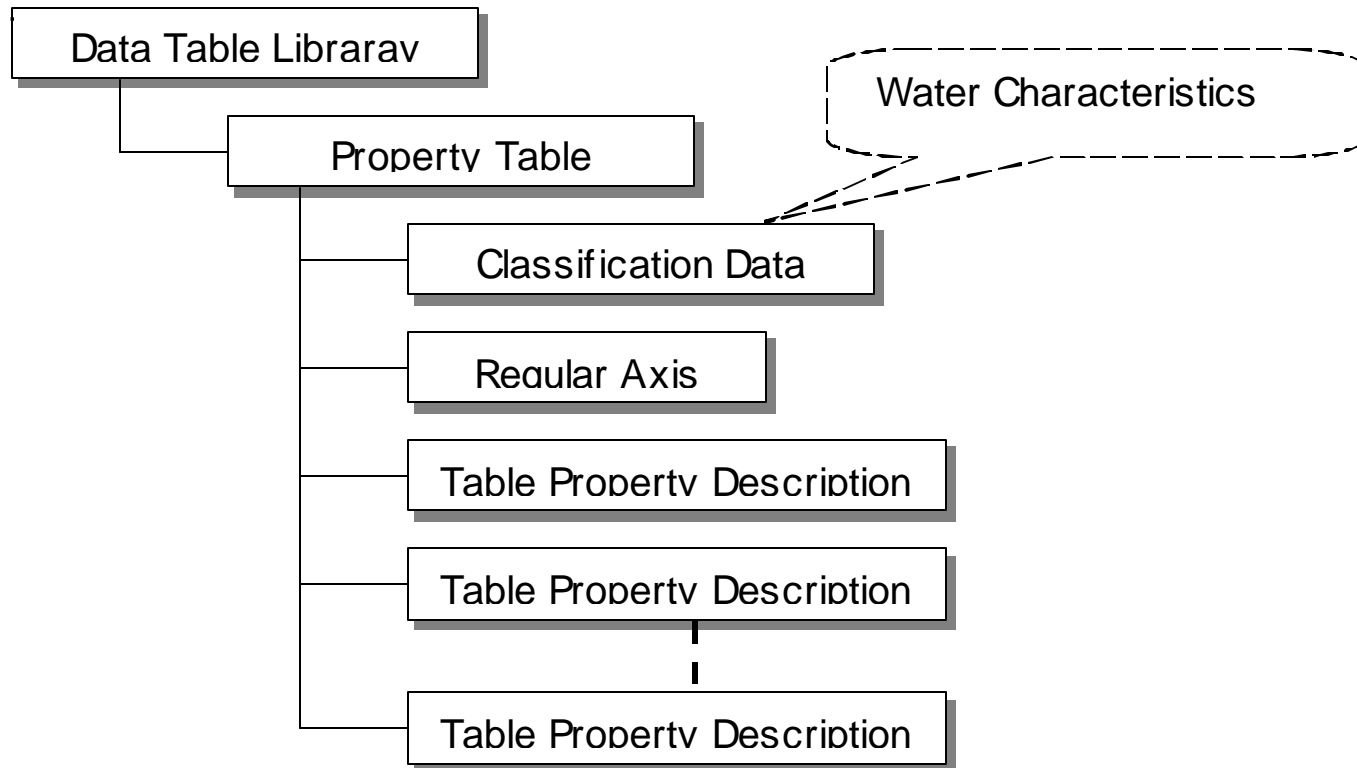
Mapping Attributes of Water Characteristics

CTDB Attribute	EAC
<i>Surface_temp</i>	EAC_WATER_BODY_SURFACE_TEMPERATURE
<i>Surf_height</i>	EAC_MAXIMUM_WAVE_HEIGHT
<i>Primary_wave.Height</i>	EAC_SIGNIF_PRIMARY_BREAKER_HEIGHT
<i>Primary_wave.Period</i>	EAC_MEAN_PRIMARY_WAVE_PERIOD
<i>Primary_wave.Direction</i>	EAC_PRIMARY_WAVE_DIRECTION
<i>Secondary wave.Height</i>	EAC_SIGNIF_SECONDARY_BREAKER_HEIGHT
<i>Secondary wave.Period</i>	EAC_MEAN_SECONDARY_WAVE_PERIOD
<i>Secondary wave.Direction</i>	EAC_SECONDARY_WAVE_DIRECTION



Mapping Attributes of Water Characteristics

[2 of 2]





Mapping Abstract Features

CTDB Abstract Feature	ECC	DRM Class
OFFROAD_SEGMENT	ECC_TERRAIN_TRANSPORTATION_ROUTE	<Linear Feature>
BRIDGE	ECC_BRIDGE_SPAN	<Areal Feature>
CRATER	ECC_TERRAIN_CRATER	<Areal Feature>
DITCH	ECC_ENGINEER_TRENCH	<Linear Feature>
<i>Hide_pos_index</i> <i>from FIGHTING_POS</i>	ECC_WEAPON_HULL_DEFILADE_POSITION	<Point Feature>
<i>Entry_pt_index from</i> <i>FIGHTING_POS</i>	ECC_ENTRANCE_AND_OR_EXIT	<Point Feature>



Mapping Crater

CTDB Attribute	EAC
<i>Force_id</i>	EAC_MILITARY_FORCE_ALLEGIANCE
<i>Crater_type</i>	EAC_TERRAIN_OBSTACLE_TYPE
<i>Avoid</i>	EAC_PASSAGE_BLOCKED
<i>Cell_id</i>	(Unsupported)
<i>Depth</i>	EAC_DEPTH_BELOW_SURFACE_LEVEL

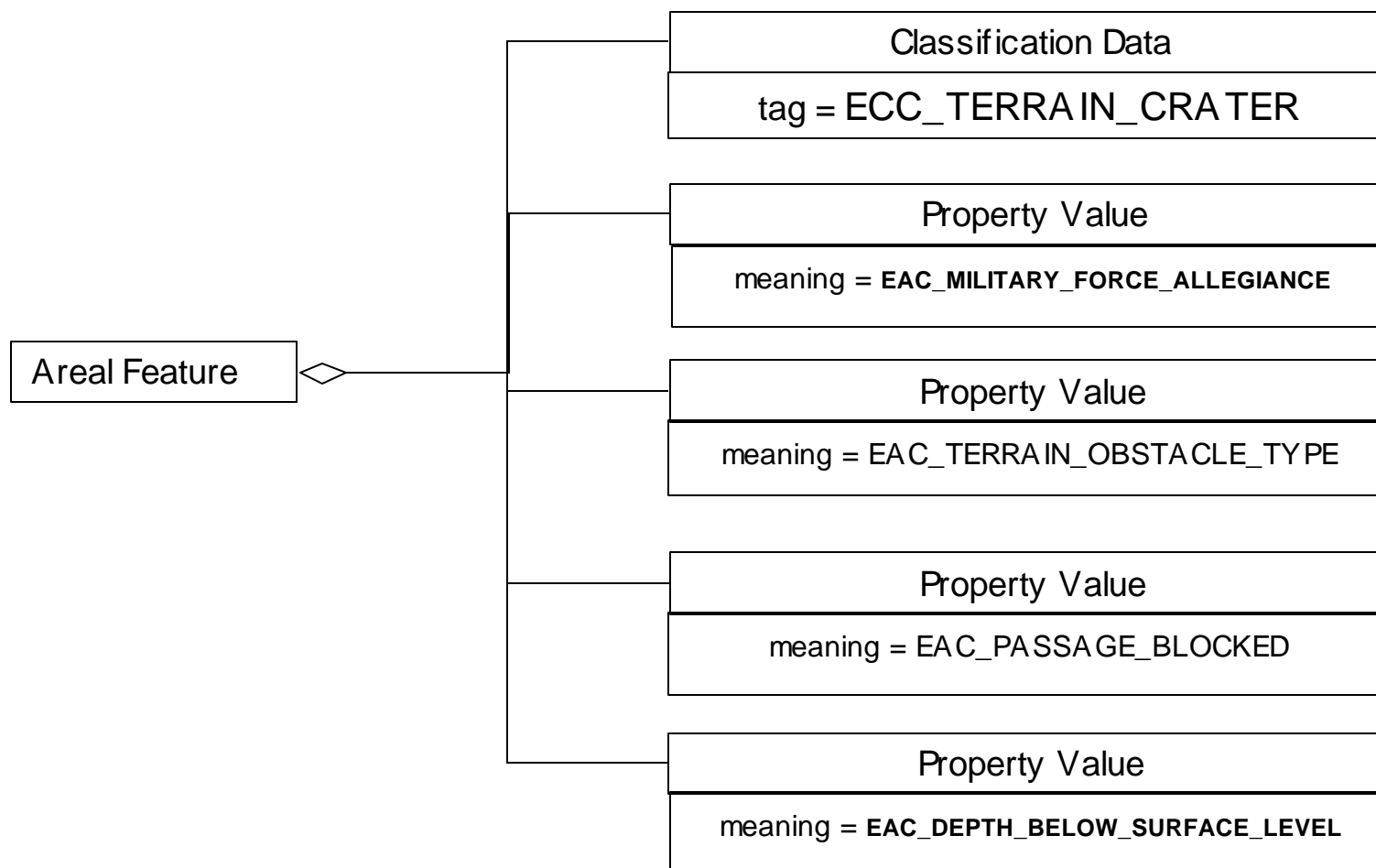


Mapping Ditch

CTDB Attribute	EAC
<i>Force_id</i>	EAC_MILITARY_FORCE_ALLEGIANCE
<i>Atd_type</i>	EAC_ENGINEER_TRENCH_TYPE
<i>Parapet</i>	EAC_HEIGHT_ABOVE_SURFACE_LEVEL
<i>Avoid</i>	EAC_PASSAGE_BLOCKED
<i>Cell_id</i>	(Unsupported)
<i>Depth</i>	EAC_DEPTH_BELOW_SURFACE_LEVEL

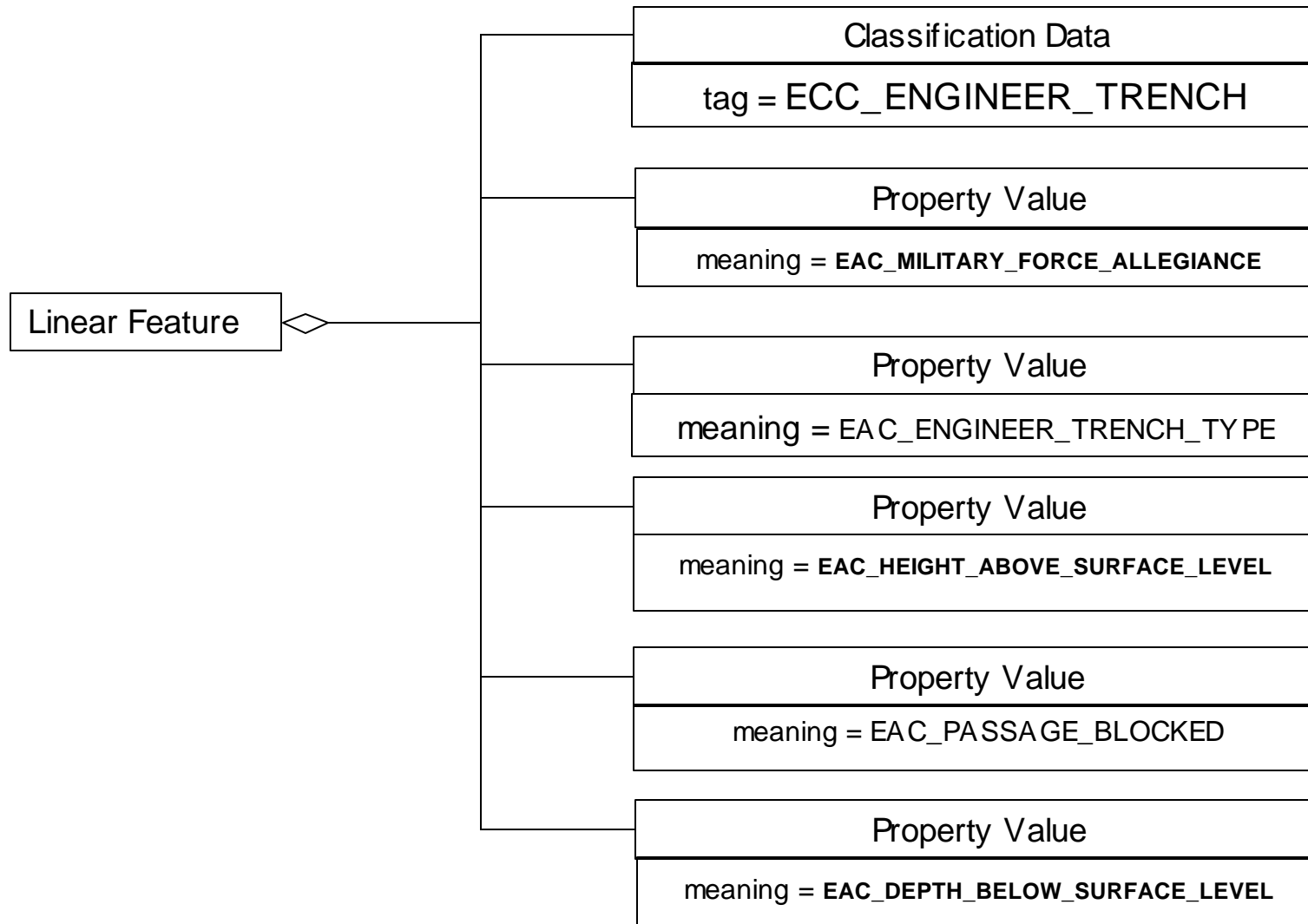


Crater Representation





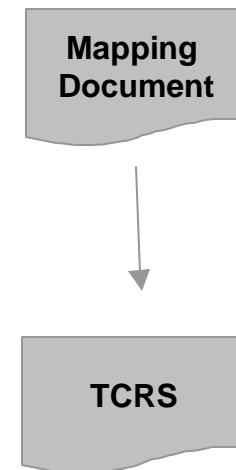
Trench Representation





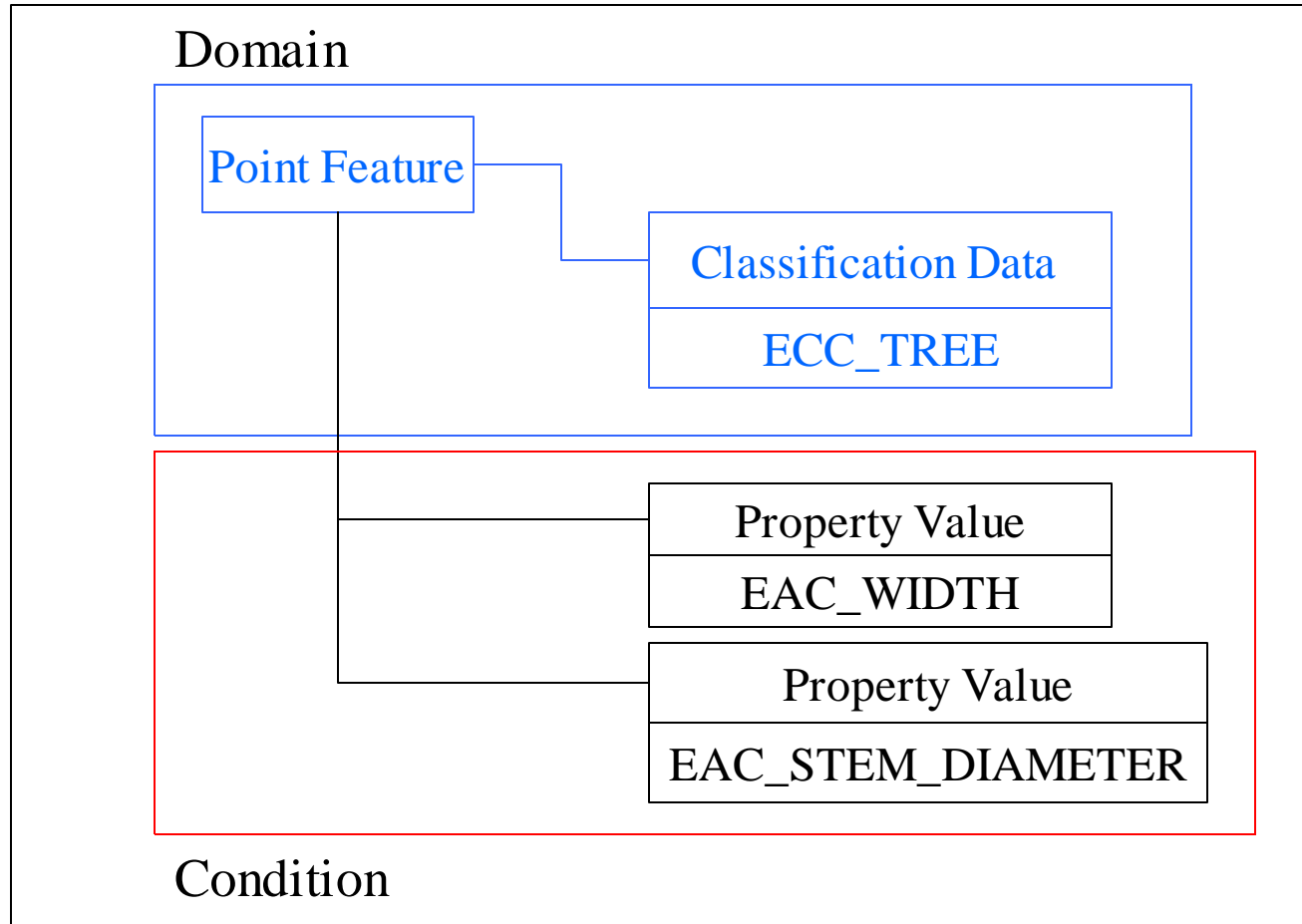
Step 3: Validation Criteria

- The mapping document provides transmittal instance diagrams that are valuable for validating the produced transmittal.
- This can be done through inspection, but that is time consuming and problematic.
- Instead, a TCRS can be derived which defines the exact requirements that must be met by the production application.
- This is done by the TCRS_Checker application which compares a transmittal to the data requirements specified in the TCRS document.
- Thus, in this step, the requirements are to be captured as exit criteria for production.
- These requirements are defined in terms of the DRM including EDCS & SRM.



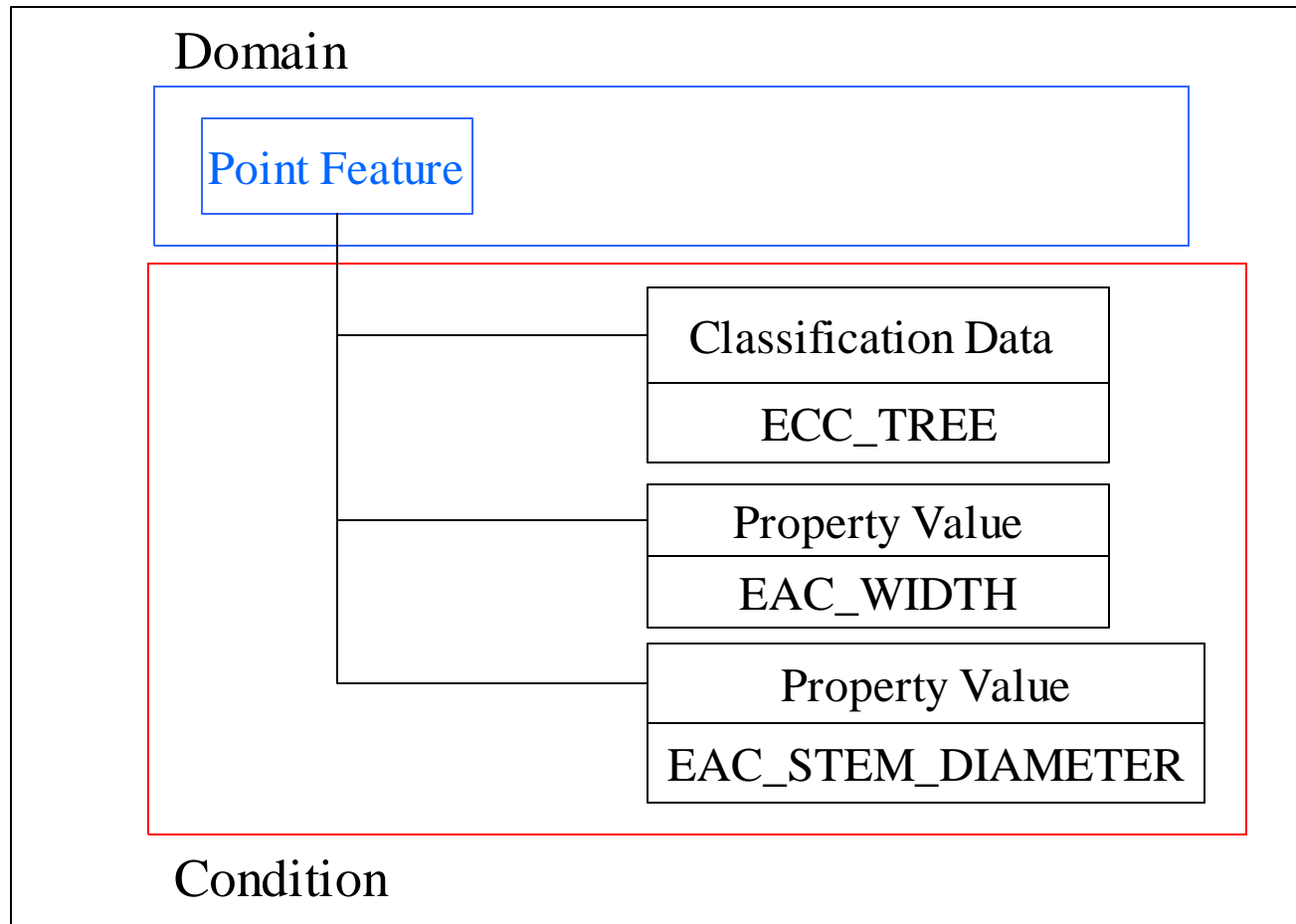


Sample Transmittal Requirement #1



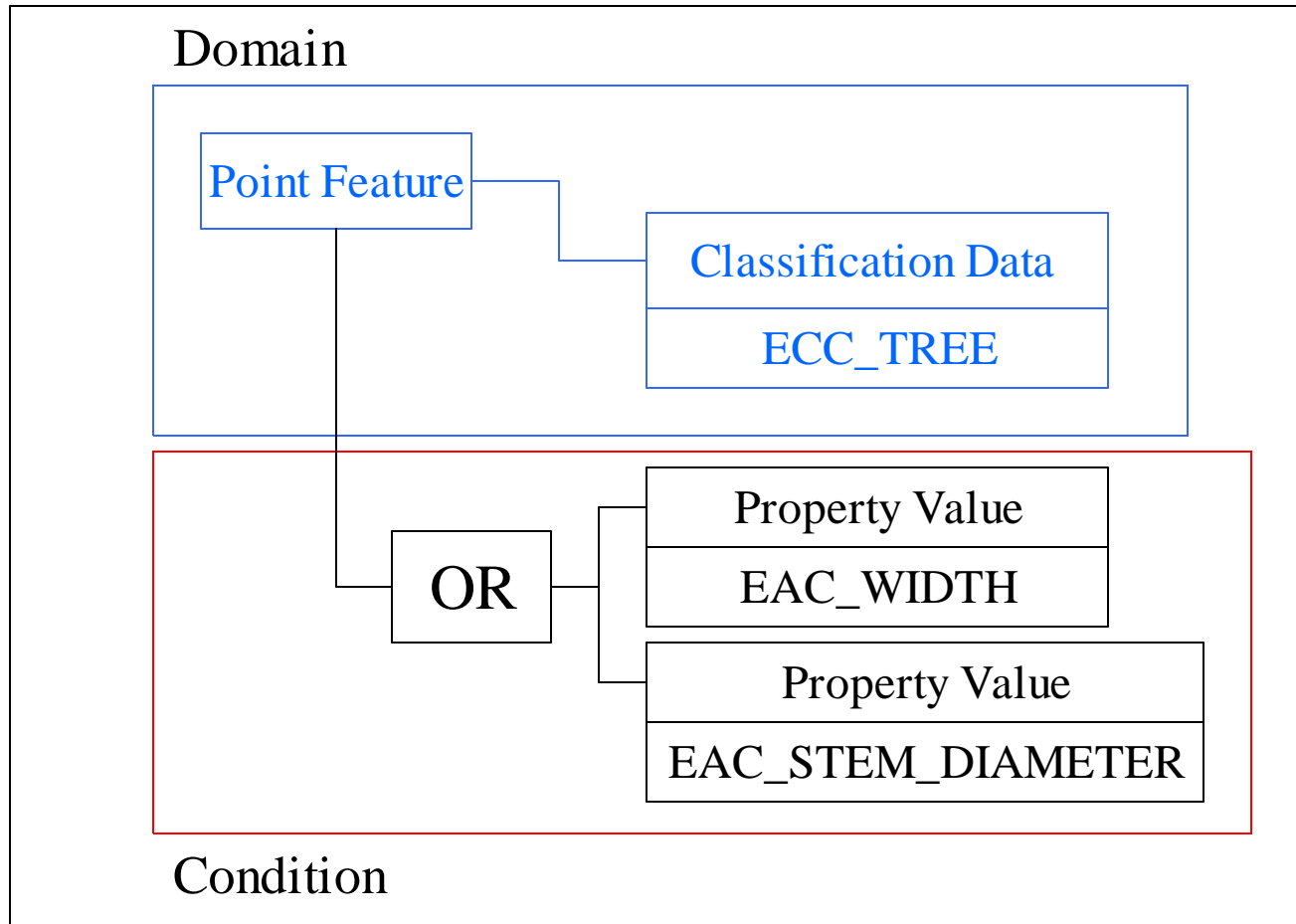


Sample Transmittal Requirement #2





Sample Transmittal Requirement #3





Step 4: Creating a Transmittal

First: Learn basic calls of the SEDRIS API: Mostly insertion functionality

Second: Get SEDRIS SDK

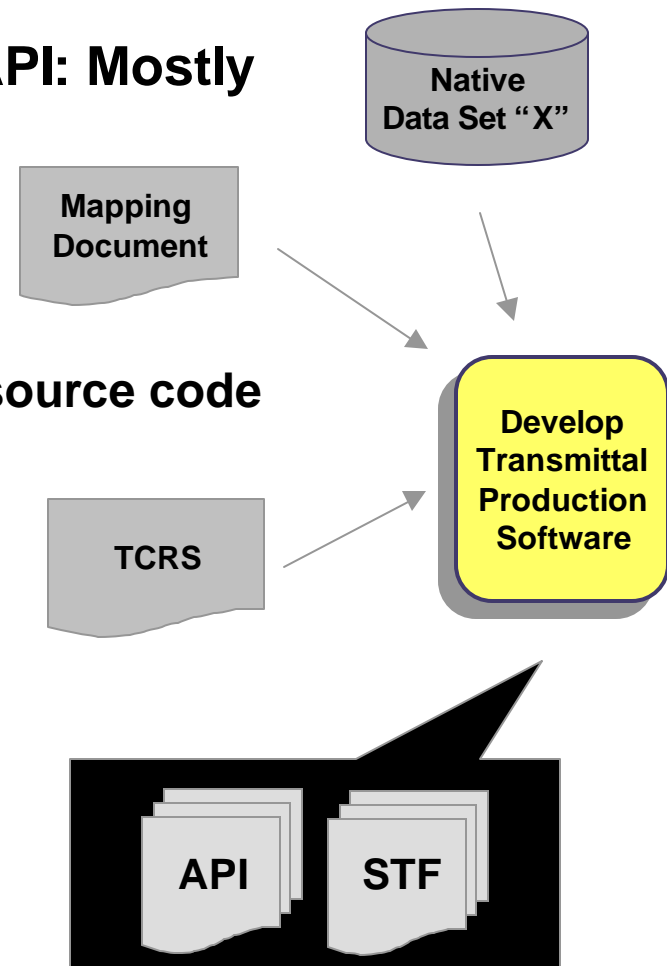
- Standard file directory structure
- Populated with the SEDRIS developer's source code distribution and sample data sets

Third: Create translating application

- Developer has full control
- Choose optimal traversal of native data
- Access to native data

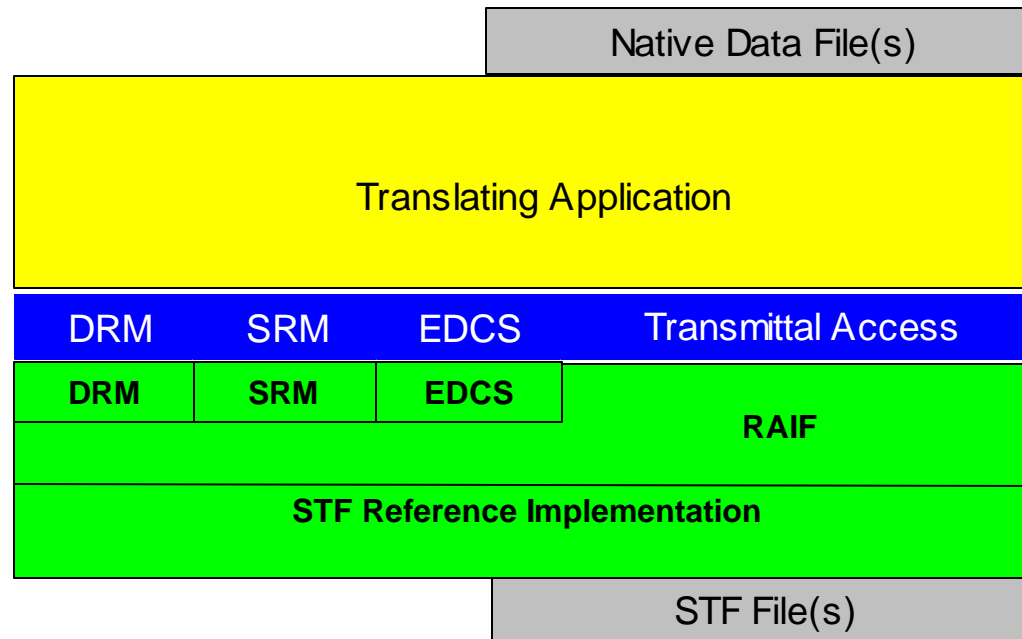
Finally: Create the STF





- Link application to core libraries
- Run Application





Step 4: Creating a Transmittal [2 of 2]



-  API Specification
-  Reference Library
-  Developer Code
-  Binary Files



Production Operations

- **Open a transmittal for writing**
- **For each mapped SEDRIIS object instance:**
 - **Create (allocate) an object of the appropriate type**
 - **Put object data (field values)**
 - **Set component, association, and link relationships**
- **Add data**
 - **<Data Table> data**
 - **<Image> data**
- **Free object memory & close the Transmittal**



Production Considerations

- **Determine if ITR should be used when producing data**
 - Will you break the transmittals into smaller transmittals?
 - Do you need incremental updates?
 - Do you want to share someone else's data?
 - Do you want to create the data in a parallel fashion?
- **To use ITR one must:**
 - Both the referencing object and the referenced object being referenced must be published through the API.
 - SE_PublishObject
 - All transmittals that contain objects using ITR must be assigned a URN name
 - SE_SetURNName



Production Considerations [2 of 3]

- **Edit Operations**
 - Overwriting object fields
 - Remove objects
 - Remove object relationships
- **Use as a post processing mechanism**
- **Example:**
 - Models with `has_units` set to false
 - Producer in Pennsylvania, user in Florida
 - Producer was made aware of issue, consumer corrected Transmittal



Production Considerations [3 of 3]

- **Object ID mechanism**
 - **SE_GetIDStringForObject()**
 - **SE_GetObjectFromIDString()**
- **Used IDs to retrieve objects already created**
 - **Exploit the sharing mechanism**
 - **Can be used to create an object without having to specify all of its relationships**
- **Provides the mechanism to map native objects to objects created through the API**



Basics of a Translating Application

- **Open Transmittal (mode=SE_AC_MODE_CREATE)**
 - **Create <Transmittal Root> and set its fields**
 - **Set the <Transmittal Root> as the root object**

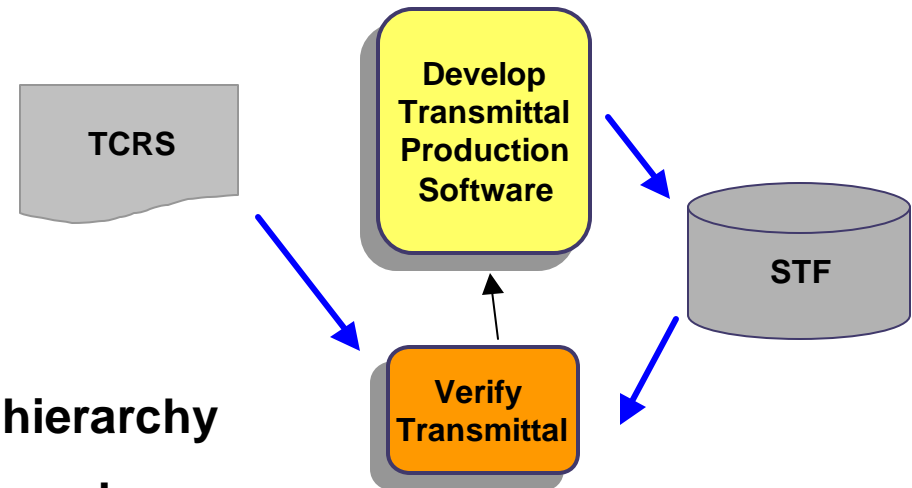
Create other objects

- **Set field values and call SE_PutFields()**
 - **If SE_PutFields() is not called will store default fields**
- **Make relationships between objects**
 - **Add components to objects**
 - **Add associations between objects**
 - **Use Object IDs to retrieve objects already created**
- **Free objects when no longer needed**
- **Close Transmittal**
 - **Handle API validation**
 - **Correct version, root object has been set**



Step 5: Validate Transmittal

- **Syntax Checker:** checks DRM compliance
- **Rules Checker:** checks DRM constraints
- **Depth**
 - traverses the entire transmittal hierarchy
 - reports statistics on types and numbers of objects encountered
- **Transmittal Browser:** View the hierarchy of a SEDRIIS transmittal
- **TCRS Checker:**
 - Validate STF with TCRS
 - Return explicit problems
- **Modify production software:** iterate until STF is valid





Questions ?

- **Questions on Producing?**
- **Documentation**
 - SEDRIS Technology Documentation Set
 - **Part 4: *Technical Reference Set***
 - **Volume 14: *How to Produce SEDRIS Transmittals***

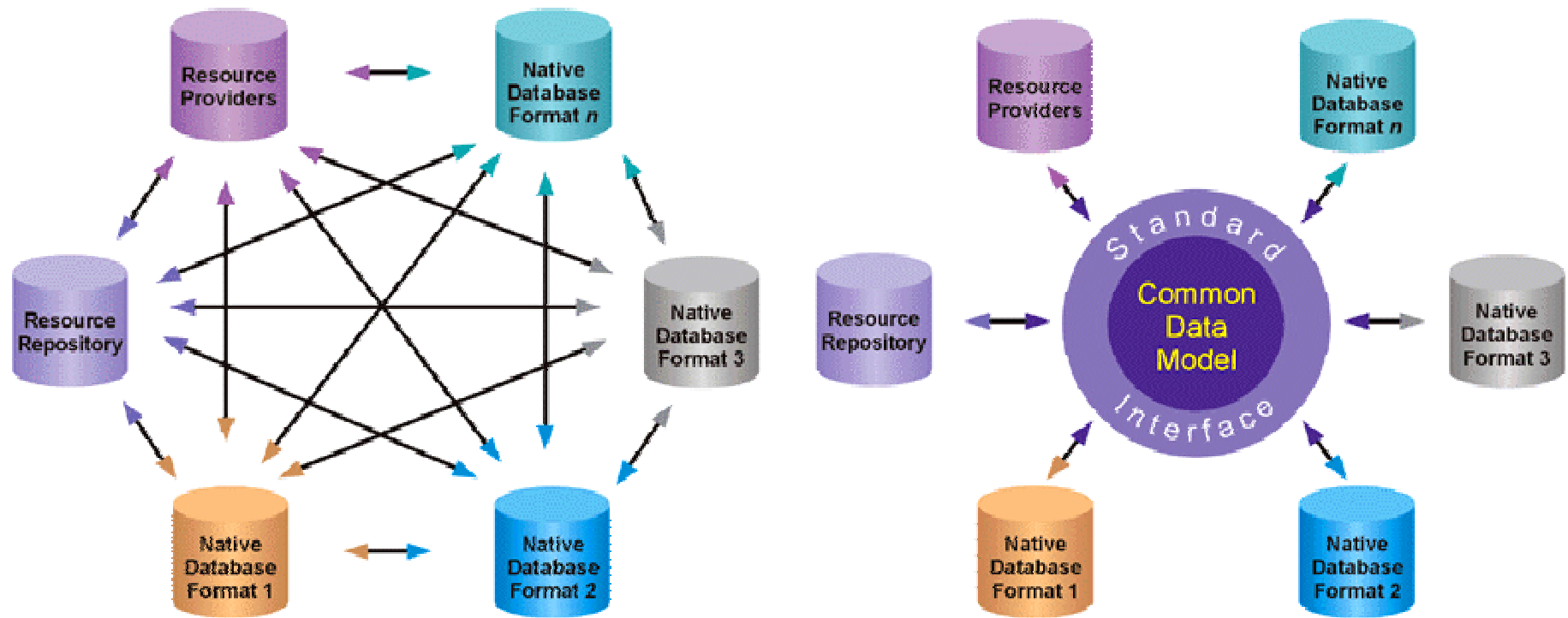


Agenda: Consuming

- **Using SEDRIIS**
- **Translating**
- **Consuming SEDRIIS**
 - **Native Analysis**
 - **Mapping Document**
 - **Extraction Capabilities**
 - **Consuming Techniques**
 - **TCRS**
 - **Consumption Strategies**
- **Consumption Challenge**
- **Solution Path**



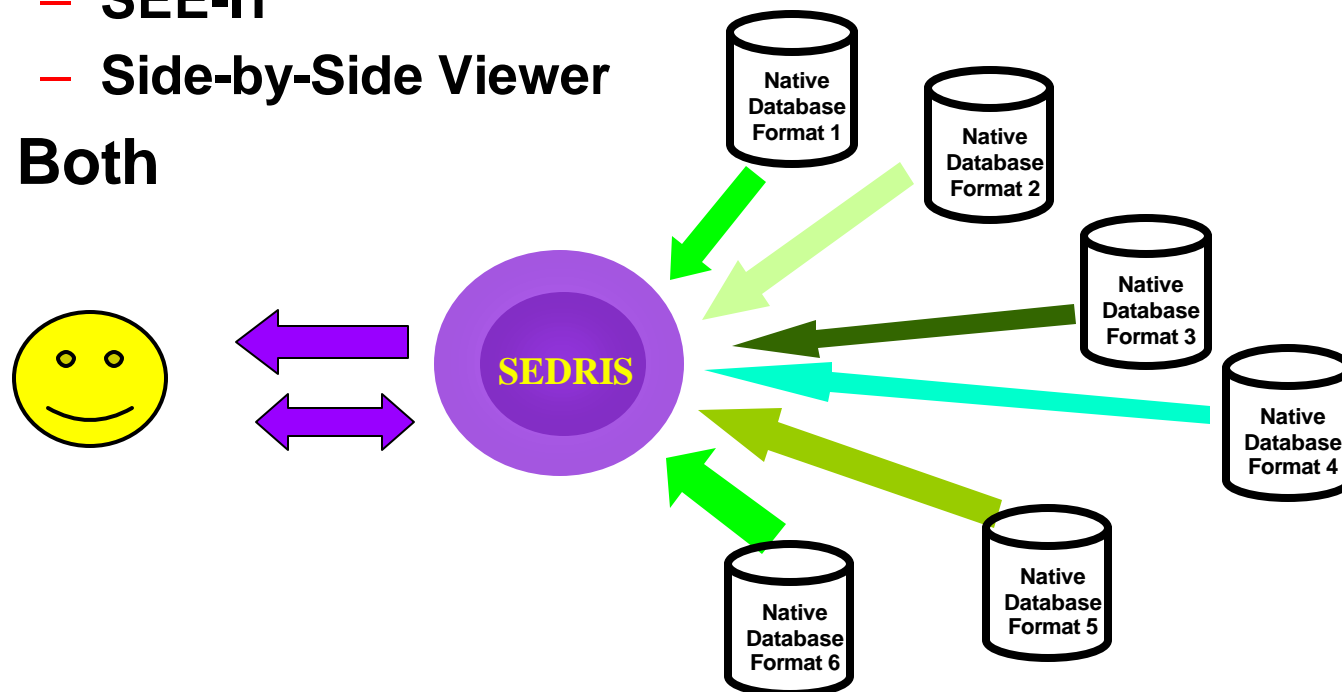
Using SEDRIS





Using SEDRI [2 of 2]

- Importing into a native format
 - STF to CTDB
 - CCTT Correlated Databases
- Importing to add value to a transmittal
 - SEE-IT
 - Side-by-Side Viewer
- Both





SEDRIIS is about ...

- **Representation of environmental data**
 - a language that *allows* articulation and expression of environmental data
- **Interchange of environmental data**
 - mechanisms to export and import data between different systems that speak the representational aspect of SEDRIIS
- **With this in mind ...**



Translating: the Language Analogy

- **Meaning is context sensitive -- Sometimes more than one context.**
- **Humans can derive the context, hence phrases such as:**
 - “The TV show has jumped the shark”
 - Parsing the words will not help, need the context
- **Machines cannot derive context**
 - Data must be unambiguous
 - Data must be loss-less
 - Data must be specific and constant
- **Likewise, the same problems appear when describing the world to automated systems**



Elements of Successful Translation

- **Parse-able syntax:**
 - *DRM*
- **Structural semantics - well-defined (or at least well-understood) grammar in the (common/foreign) language**
 - *SEDRI: DRM semantics*
- **Reasonable mapping of input language to output language possibilities**
 - *if the languages don't match in expressive or representation power, stop, we have a problem!*
- **Data's organizational semantics**
 - *e.g. Airport: point feature; areal feature; bag of polygons; spatially-organized bags of polygons; bag of "objects"; spatially-organized bags of "objects", ...*



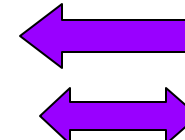
Elements of Successful Translation [2 of 2]

- **Context & understanding, or at least knowledge of context**
- **“Translator’s” command of both languages**
 - *how good is the algorithm/logic for translation and how well it’s implemented*
- **Use of “translation services”**
 - *e.g. API, filters, ...*
- **Post-translation value-adding**
 - *e.g. turn polygons for use by z-buffer; undo SRF translation*
- **Post-translation tailoring**
 - *e.g. tune polygons for runtime processing*



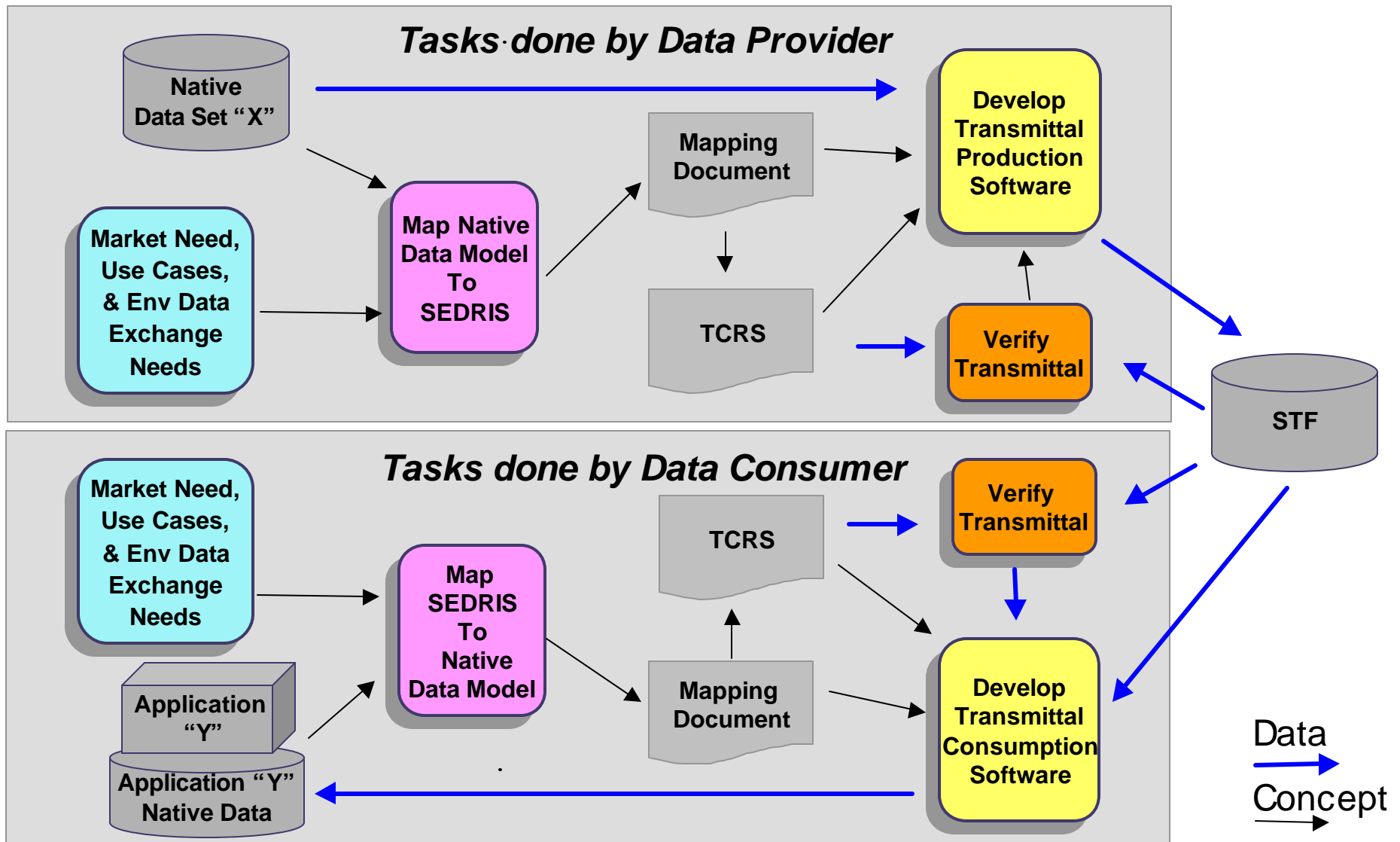
Needs & Expectations

- **Common Language - DRM**
 - Proper syntax
 - Expressive grammar
 - Dictionary of terms
 - Meta-data for structural semantics, organizational semantics, context semantics
- **Common Dictionary – EDCS**
 - Classification i.e. “things”
 - Attributes – “adverbs & adjectives”
- **Common Understanding – SRM**
 - Framework to convert data elements
- **Common Services – API**
 - Access all “language” elements
 - Provide filtering (common & controllable)
 - Provide SRM operations
 - Common “value-adding” modules





Production and Consumption Process





Consuming SEDRI

- **Step 1: Native Analysis**
 - What are your data requirements?
- **Step 2: Mapping Document**
 - Map the DRM to your data requirements
- **Step 3: Learn Implementation Details**
 - Extraction capabilities
 - Detailed DRM knowledge
- **Step 4: Determine consumption philosophy**
 - TCRS
- **Step 5: Create consumption software**
 - Taking advantage of the common services
- **Step 6: Expand consumption base**
 - Reorganizing software



Step 1: Native Analysis

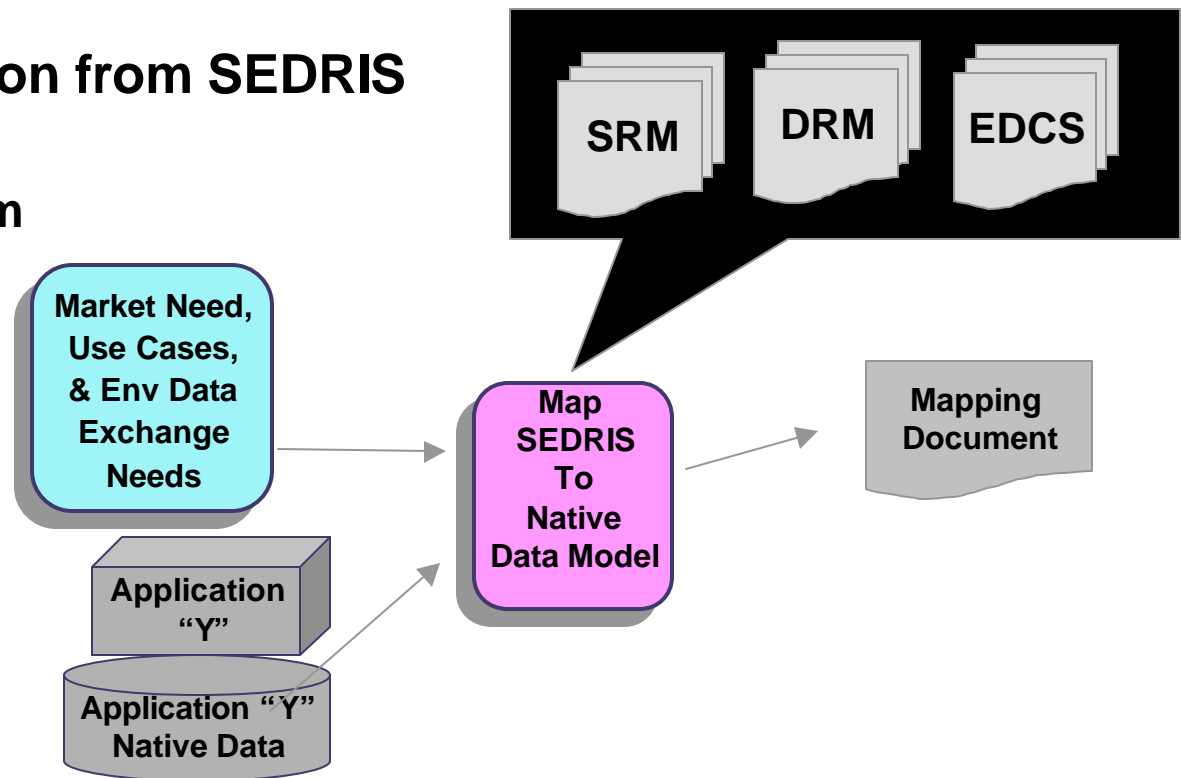
- **Analyze your data requirements**
- **If you are a tool maker, what does your tool require?**
- **If you want to import into a native format:**
 - *What type of data does your native format contain?*
 - *Map your native data model to the SEDRIS DRM*
- **The SDRM is flexible (and the EDCS is complex)**
 - *How much flexibility will you allow?*
 - *How much complexity can you handle?*

Market Need,
Use Cases,
& Env Data
Exchange
Needs



Step 2: Mapping Document

- Describe the native primitive data: What is the native organization?
- Describe the translation from SEDRI to native data/format
 - What data maps from SEDRI to native data model?
 - What data do you need to derive?
 - What algorithm will be used to derive native information?
 - What data are you going to ignore?
- Any run time data requirements?
- Focus on primitive data





Example: DRM to CTDB Mapping

- **CTDB: Volume Models (buildings, obstacles)**
 - Stored in CTDB as a set of 3D vertices representing the roofline of the volume.
 - SAF “drops down” walls to the terrain for each pair of vertices.
 - CTDB limits number of vertices to 15.

- **DRM**
 - **<Areal Feature>**.
 - With or without a <Property Value> to indicate the height above terrain
 - **<Union of Primitive Geometry>**
 - Classified with appropriate classification code
 - What is appropriate – check the EDCS query tool
 - **<Point Feature>**
 - With or without <Property Value> to indicate height, width, and orientation
 - With appropriate classification



DRM to CTDB Mapping [2 of 3]

- CTDB –
 - Tree lines
 - Stored as a linear set of points
 - Trunk Radius
 - Foliage Height
 - Fullness
 - Total Height
 - Affects intervisibility and acts as an obstacle to vehicle movement
 - Individual Trees
 - Stored as tree line with only one point
- DRM
 - <Point Feature> classified as a tree
 - <Linear Feature> classified as a tree line
 - <Union of Primitive Geometry> classified as a tree or tree line
 - <Areal Feature> classified as a tree



DRM to CTDB Mapping [3 of 3]

- Tree Canopies
 - CTDB
 - Stored as a set of polygons representing the roof of the canopy
 - The fullness of the canopy is also stored to represent the density of the forest area
 - Canopy polygons are used in intervisibility calculations
 - DRM
 - <Polygons> properly attributed
 - <Areal Feature> properly attributed
 - <Linear Feature> properly attributed
- Railroads and Pipelines
 - CTDB
 - Both stored as linear sets of points
 - DRM
 - <Point Feature>
 - <Geometry>
 - What is the complete classification?



Step 3: Learn Implementation Details

- **Extraction Capabilities**
 - Tree traversal
 - Selection and filtering of objects
 - Spatial boundaries
 - Search filters
 - Automatic navigation via branching criteria
 - Retrieval of entire object hierarchy
 - Automatic transformation of coordinates & colors
 - Object Identification
 - Advanced DRM Functionality
 - ITR
- **DRM requirements**



Basic Extraction

<u>Description</u>	<u>SEDRI API</u>
Open a Transmittal	SE_OpenTransmittalByFile
Get the root object	SE_GetRootObject
Set the current SRF Parameters	SE_SetSRFParameters
Set the Color Model	SE_SetColorModel
Create a Search Boundary	SE_CreateSpatialSearchBoundary
Create a Search Filter	SE_CreateSearchFilter
Create an Iterator	SE_InitializeComponentIterator
Retrieve an object	SE_GetNextObject
Access and process object attributes	
Free the object	SE_FreeObject
Free the Iterator	SE_FreeIterator
Free the Search Filter	SE_FreeSearchFilter
Repeat for as many different Search Filters as desired	
Free the Search Boundary	SE_FreeSpatialSearchBoundary
Close the Transmittal	SE_CloseTransmittal



Detailed DRM Knowledge

- **Primitive' objects**
 - **Features**
 - <Point Features>, <Linear Features>, <Areal Features>
 - **Geometry**
 - <Camera Points>
 - <Points>, <Lines>, <Arcs>, <Ellipses>, <Elliptic Cylinders>
 - <Polygons>, <Finite Element Meshes>
 - <Light Sources>
 - <Sounds>
 - <Property Tables>, <Property Grids>
 - <Images>
 - **Model Instances**
- **Modifiers, Attributes, & Components**
 - <Location>
 - <Color>
 - **Classification**
 - **Transformation**

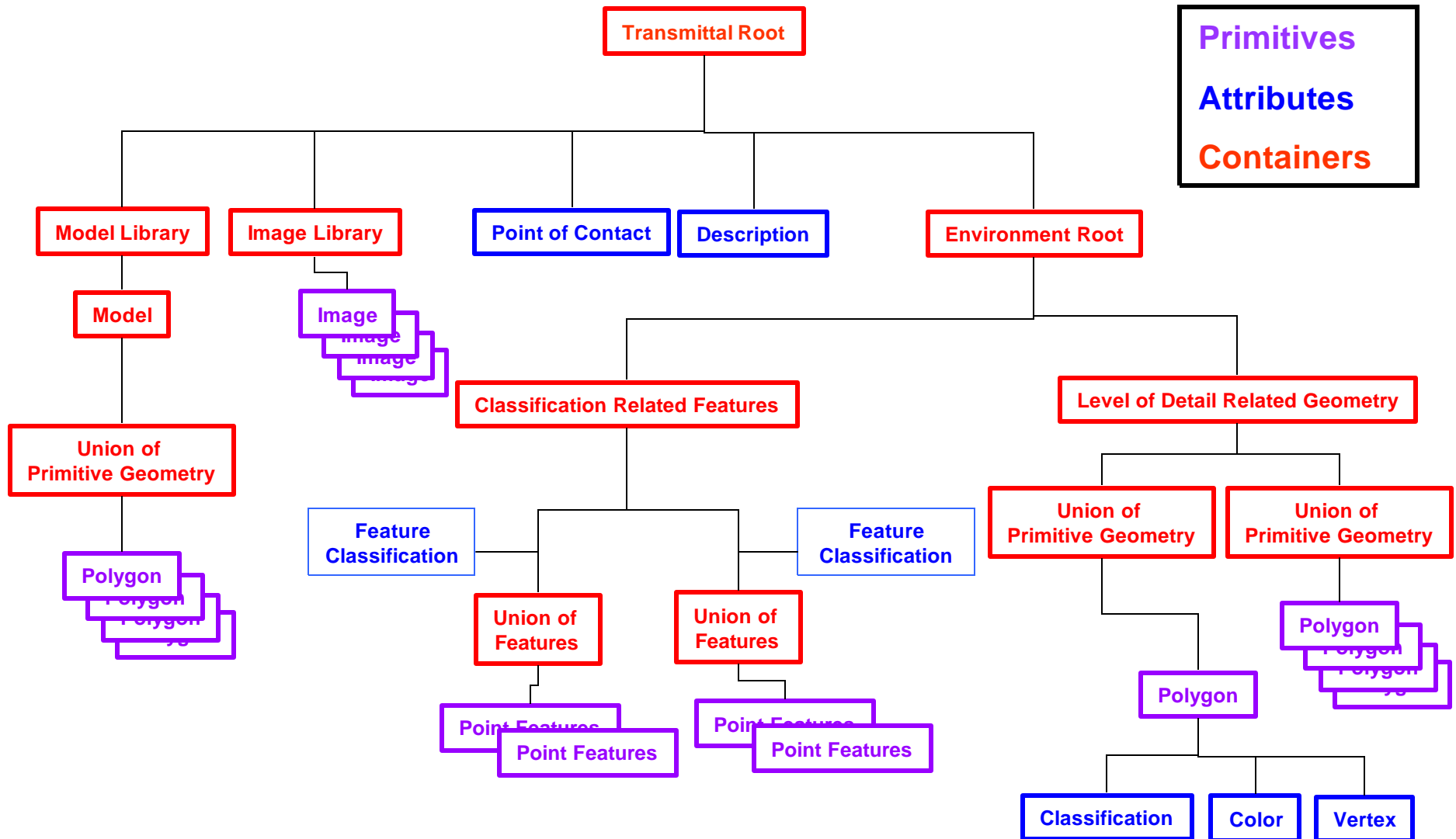


Detailed DRM Knowledge [2 of 3]

- **Containers and Organizers**
 - **<Transmittal Root> and <Environment Root>**
 - **Libraries**
 - **<Model>, <Image>, <Data Table>, <Property Set>, <Colour Table>, <Symbol>, <Sound>**
 - **Feature & Geometry Hierarchies**
 - **Classification (ECC)**
 - **Scale (ESC)**
 - **Spatial Index**
 - **Perimeter**
 - **Time**
 - **Level of Detail (LoD)**
 - **Quad Tree & Oct Tree**
 - **Alternate Hierarchy**
 - **Separating Plane (G)**
 - **Continuous LoD (G)**
 - **Animation (G)**
 - **Union**



Detailed DRM Knowledge [3 of 3]





Step 4: Consumption Philosophy

- **The mechanics of data extraction is the easy part ...**
- **By design, the SEDRIIS API tackles ease-of-use issues**
 - Filtering is great, but what do you want to filter on/for?
- **Before you start, you need a plan**
 - In particular, you need to be prepared to handle:
 - **Your ideal transmittal**
 - The one that you'd build, if you could ...
 - **Your acceptable transmittal**
 - The one that you could imagine someone building ...
 - **Your worst-case transmittal**
 - The one you can't imagine that anyone in their right mind ...
 - **More transmittals than you might like will be non-ideal**
 - You'll need a plan ... because you might not get what you want
- **Be Prepared! Do your systems analysis, mapping, and design**

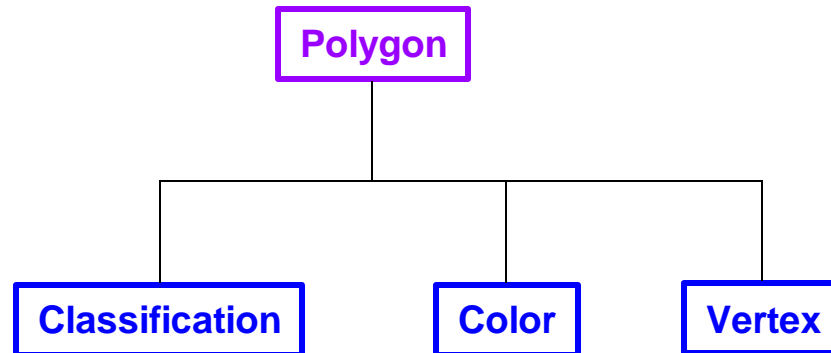


Step 4: Consumption Philosophy [2 of 4]

- **Design and implement a data consumption strategy which focuses on a narrow band of acceptable transmittal data models?**
 - Reduces cost, but limits flexibility
 - E.g., can I accept elevation grids instead of polygons?
 - May not be scalable to other “nonstandard” data providers
 - E.g., use NGA DTED instead of a prepared TIN?
- **Design and implement a data consumption strategy which accepts a wider range of transmittal data models?**
 - Increases early costs, but enhances flexibility
 - Should be scalable, given good design choices
- **Goal: Design for the future, implement on the pay-as-you-go plan**



Step 4: Consumption Philosophy [3 of 4]



- **What are you consuming?**
 - Primitives with attributes/modifiers
- **Find the least common denominator**
 - In SEDRIS terms
- **Removes the organization issue**



Step 4: Consumption Philosophy [4 of 4]

- **Transmittal Content Requirements Specification (TCRS)**
 - Specification of what your software will minimally handle
 - Provides different DRM representations that you will and will not handle
 - Defines the requirements of a Transmittal
 - Allows you to evaluate Transmittal content
 - NOT a Content Agreement
 - Flows from the Mapping Document
- **TCRS Example**
 - CATT CDB
 - STF to CTDB



CATT TCRS Example [1 of 4]

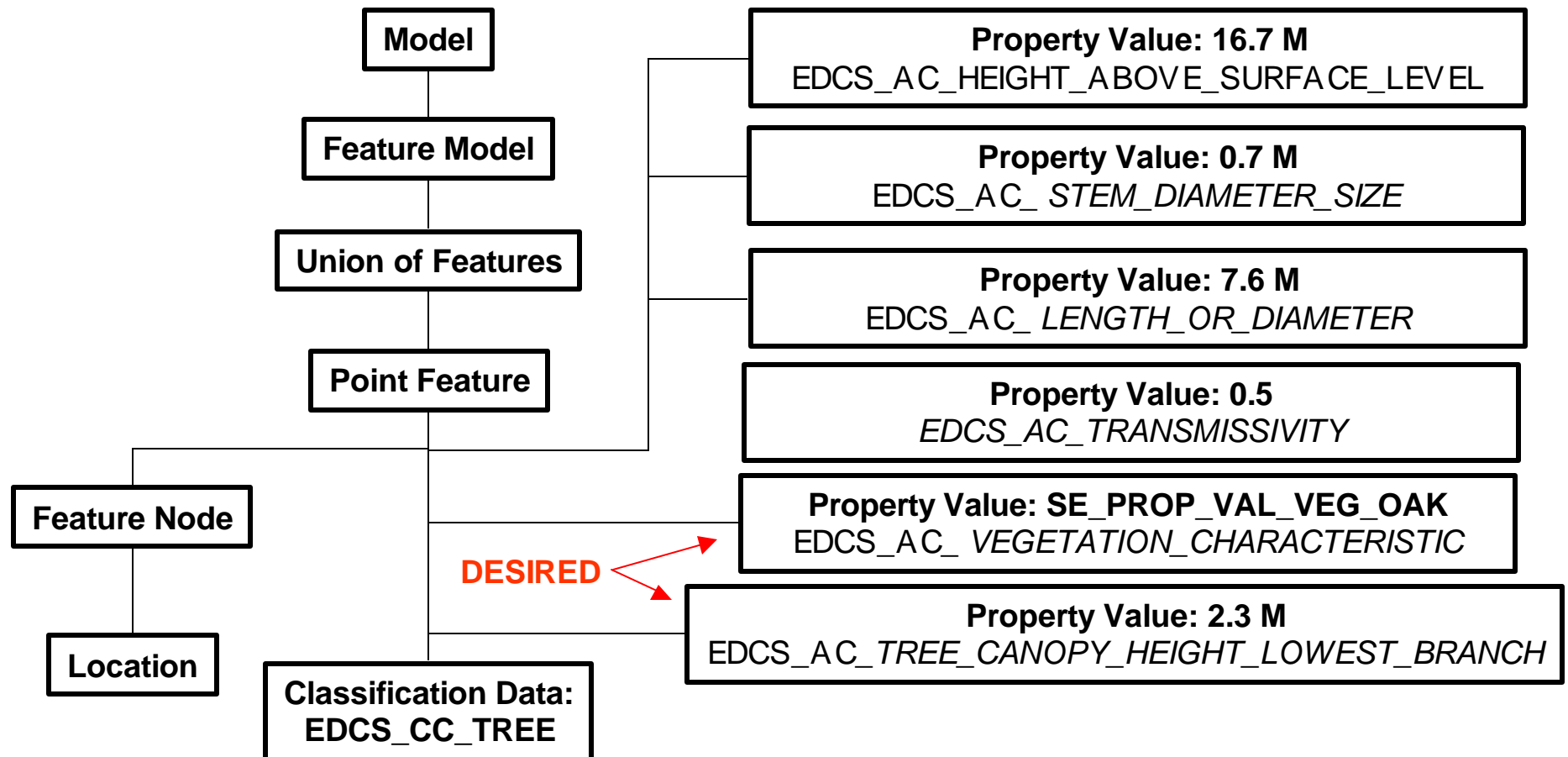
- **Section 3.1: Tree/Bush Models**

- **Structure: Model ✂ Feature Model ✂ Union of Features ✂ a single Point Feature ✂ Feature Node ✂ Location**
- **Requirements:**
 - **A <Classification Data> component identifying model as a tree or bush:**
EDCS_CC_TREE or EDCS_CC_SCRUB_OR_BRUSH_OR_BUSH
 - **A <Property Value> component specifying the height from ground to top of foliage: attribute_code = EDCS_AC_HEIGHT_ABOVE_SURFACE_LEVEL in meters and Float 32**
 - **A <Property Value> component specifying the trunk diameter:**
EDCS_AC_STEM_DIAMETER_SIZE in meters and Float 32
 - **A <Property Value> component specifying the foliage diameter: attribute_code = EDCS_AC_LENGTH_OR_DIAMETER, in meters and Float 32**
 - **A <Property Value> component specifying the foliage transmissivity between 0.0 and 1.0: attribute code = EDCS_AC_TRANSMISSIVITY, in percent and Float 32**
 - **A <Property Value> component specifying the tree/bush type: attribute code = EDCS_AC_VEGETATION_CHARACTERISTIC, an enumerator of Unsigned Int 16**
 - **A <Property Value> component specifying the distance from the ground to the foliage: attribute code = EDCS_AC_TREE_CANOPY_HEIGHT_LOWEST_BRANCH, in meters and Float 32**



CATT TCRS Example [2 of 4]

- Section 3.1: Tree/Bush Models





CATT TCRS Example [3 of 4]

- **Section 4. Terrain Skin**
 - A continuous surface covering the extents of the database
 - CATT stores the data as a two dimensional data table of elevation values
 - SEDRIS supports the concept of a terrain skin by allowing items to be identified as “terrain”. Grids and other SEDRIS objects can be identified as “terrain” through the use of ECCs
 - 2 SEDRIS constructs
 - <Polygons> classified as terrain polygons
 - <Property Grids> with elevation and diagonalization issues
 - So which did CATT choose?



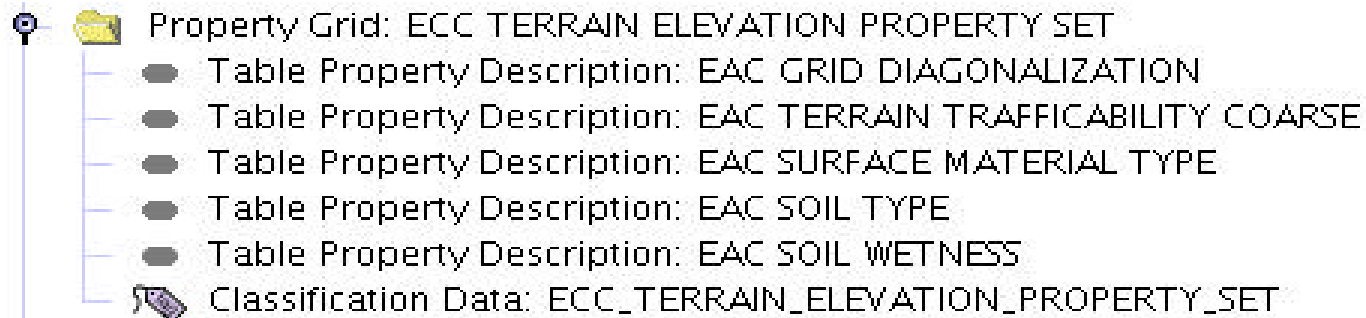
CATT TCRS Example [4 of 4]

- **BOTH**
 - CATT will consume a <Property Grid>
 - Using a polygonal representation <Polygons> will have:
 - An ECC of EDCS_CC_TERRAIN
 - The SE_TERRAIN_POLYGON value set within the polygon_flags field of the Polygon
 - If the <Polygon> is a “cut” polygon, the SE_CUT_POLYGON value set within the polygon_flags field
 - If the <Polygon> is a “raised” or “fill” polygon, the SET_RAISED_POLYGON value set with the polygon_flags field
 - A <Property Value> of EDCS_AC_SURFACE_TRAFFICABILITY_GROUP_CCTT
 - For polygonal representation will include an empty <Property Grid>

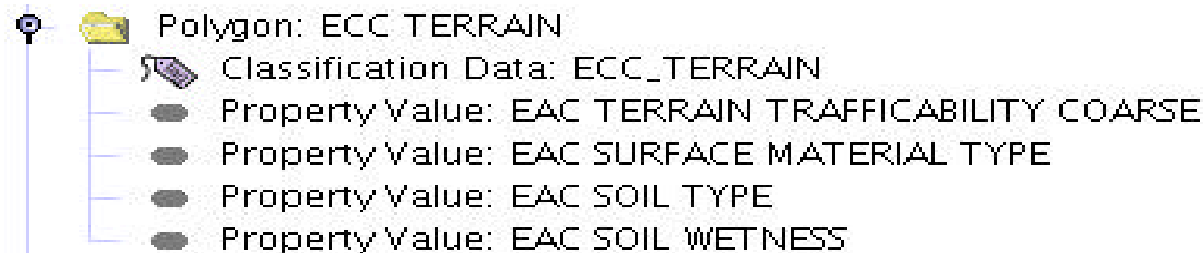


Example: STF to CTDB TCRS

- **Limit terrain to:**



- **OR:**



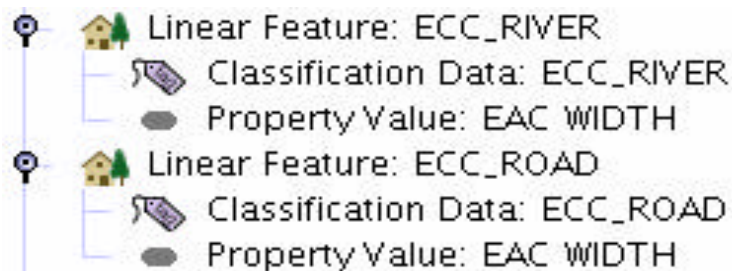


STF to CTDB TCRS [2 of 4]

- **Buildings shall be:**



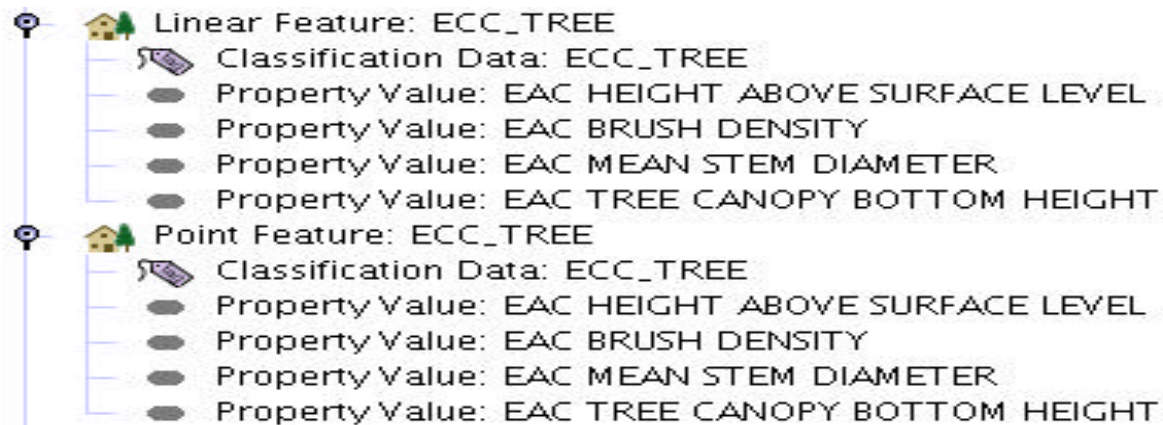
- **Roads and rivers:**



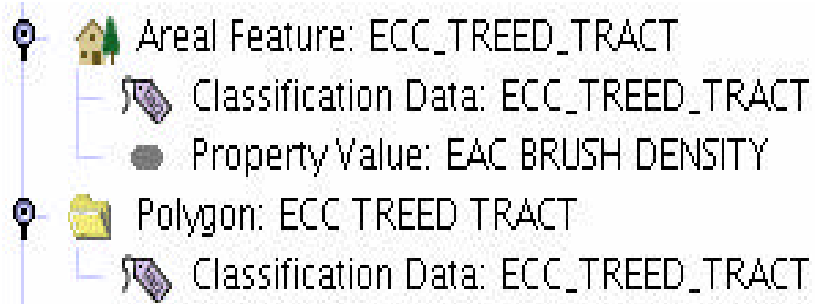


STF to CTDB TCRS [3 of 4]

- **Trees & tree lines:**












- **Tree Canopies**












STF to CTDB TCRS [4 of 4]

- **Railroad, power lines, & pipeline**

-   Linear Feature: ECC_RAILROAD
 -  Classification Data: ECC_RAILROAD
-   Linear Feature: ECC_POWER_TRANSMISSION_LINE
 -  Classification Data: ECC_POWER_TRANSMISSION_LINE
-   Linear Feature: ECC_PIPELINE
 -  Classification Data: ECC_PIPELINE

- **Soil information**

-   Areal Feature: ECC_GROUND_SURFACE_ELEMENT
 -  Classification Data: ECC_GROUND_SURFACE_ELEMENT
 -  Property Value: EAC TERRAIN TRAFFICABILITY COARSE
 -  Property Value: EAC SURFACE MATERIAL TYPE
 -  Property Value: EAC SOIL TYPE
 -  Property Value: EAC SOIL WETNESS



Step 5: Consumption Software

- Must reflect (and implement) the input to output mapping requirements
- Must rely on the input and the output languages' syntax and semantics (as opposed to implied semantics)
- Must not make assumptions that are not supported in the input or the output language
- Must not hard-wire extra-transmittal context
- Should use the common services to fullest extent
- Must be able (and willing) to process enough data to extract the information it needs

Develop
Transmittal
Consumption
Software



Consumption/translation strategies

- **Implement a method for each class in SEDRI (e.g., if a SIRG is found, do <action>)**
- **Use filters (API) to look for desired data**
- **Configure extraction approach based on syntax and/or semantics data extracted from transmittal or pre-processing of transmittal**
- **Combination of the above**
- **Use meta-data and summary information**



Extraction Techniques

- If you must transform SRFs or color systems, let the API do it
 - It's probably more efficient and more exact than what you may be using currently
- Extract each <Library>, or the <Environment Root>, first?
 - Get 'em all, even if not used vs. Get 'em as they are referenced
- Four basic <Environment Root> extraction strategies
 - *Get it all, everywhere*
 - Depth-first traversal of entire transmittal
 - “walk the tree” and examine everything at application-level
 - *Get it all, somewhere*
 - *Get some of it, everywhere*
 - *Get some of it, somewhere*
- Choice driven by application-requirements



Making the API work for you

```
extern SE_STATUS_CODE_ENUM
SE_InitializeComponentIterator(
    SE_OBJECT                start_object,
    SE_SEARCH_BOUNDARY       boundary,
    SE_SEARCH_FILTER         filter,
    SE_BOOLEAN               directly_attach_table_components,
    SE_BOOLEAN               process_inheritance,
    SE_BOOLEAN               transform_locations,
    SE_BOOLEAN               follow_model_instances,
    SE_BOOLEAN               evaluate_static_control_links,
    const SE_HIERARCHY_SELECT_PARAMETERS *select_parameters_ptr,
    const SE_HIERARCHY_ORDER_PARAMETERS *traversal_order_parameters_ptr,
    SE_GENERAL_TRAVERSAL_ENUM general_traversal_pattern,
    SE_ITR_TRAVERSAL_ENUM    inter_transmittal_referencing_traversal,
    SE_ITERATOR              *iterator_out_ptr);
```

- Use as much of the advanced features as you need!



Making the API work for you [2 of 13]

- Retrieve based on locations
- Search Type
 - Point: The center point of the object tested
 - Bounding Box: A box around the object is tested
 - Exact: The geometry of the object is tested
- Inclusion
 - Full Inclusion: Objects must be fully contained within the search bounds
 - Partial Inclusion: Objects must only be partially contained
- Search box
 - Define a min and max point in the SRM of the <Environment Root>
- Search dimensionality
 - 2-D or 3-D
 - 2-D allows infinite height/depth



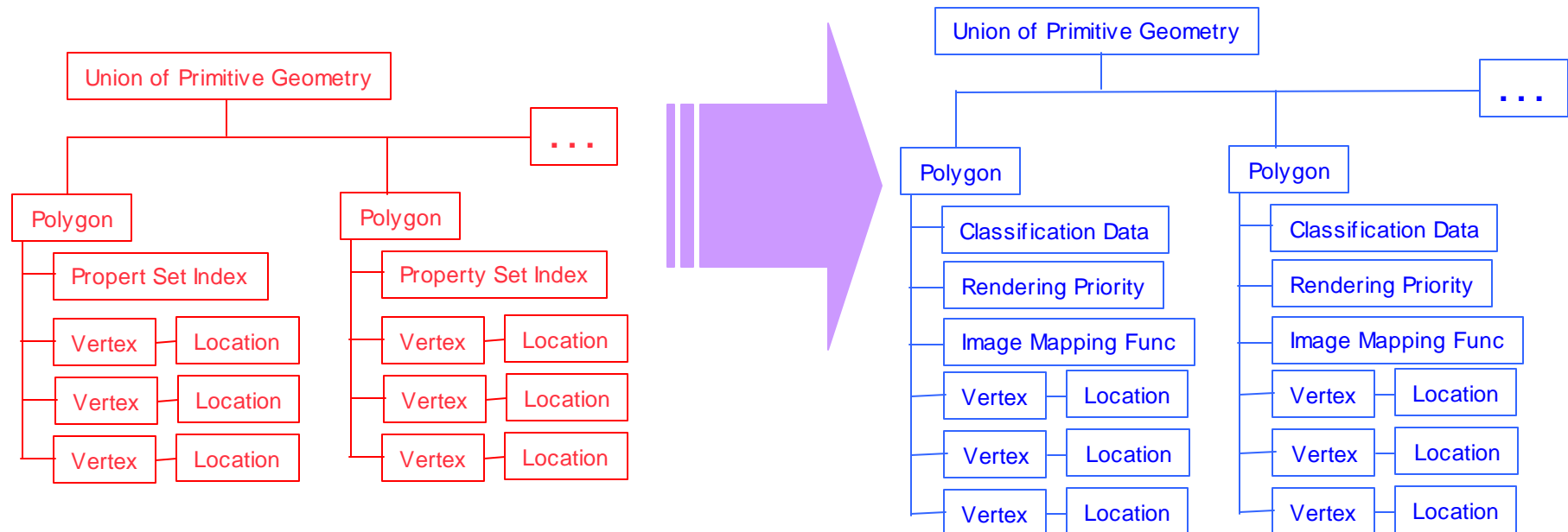
Making the API work for you [3 of 13]

- **Limiting the search through the search filters**
- **Get only:**
 - Objects of specific DRM classes
 - Objects that have associate objects
 - Objects that have associated objects of specific DRM class
 - Objects with components of specific DRM classes
 - Objects with components with specific field values
 - Objects with specific field values
 - Objects with specific field ranges
 - Objects within X levels down the tree
 - Objects that meet my specified function
 - Combination of these items



Making the API work for you [4 of 13]

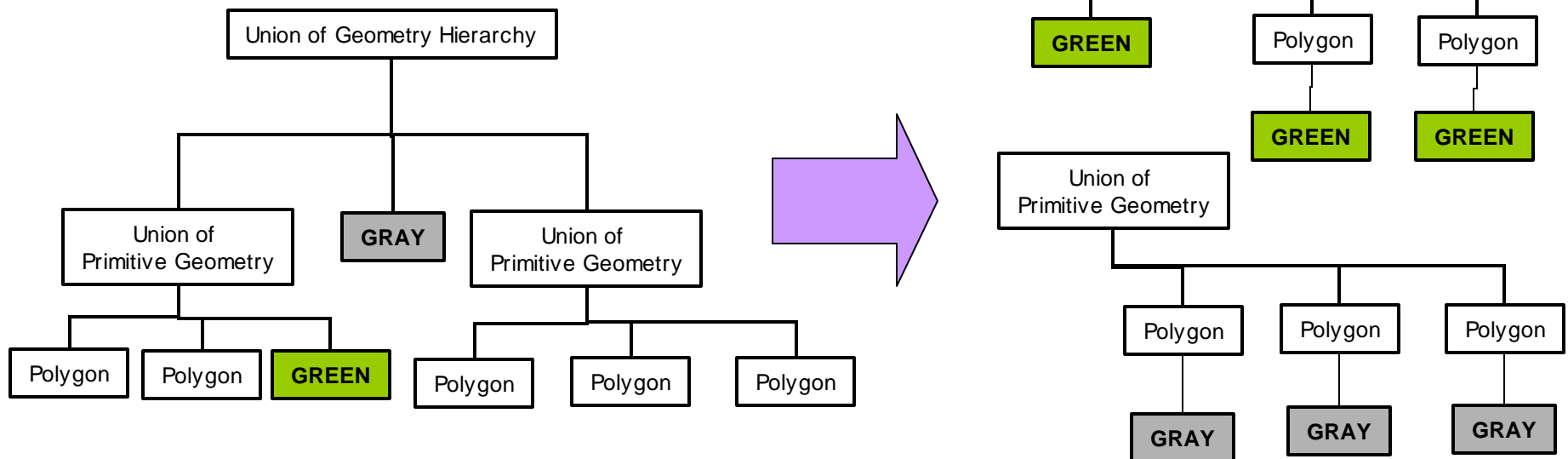
- **directly_attach_table_components**
 - When set to SE_TRUE, indexed components are replaced with directly attached instances (right example)
 - Examples: <Attribute Set Index> objects, <Color Index> , ...
- **Let the API do it**
 - Simplifies consumption
 - Work with primitives





Making the API work for you [5 of 13]

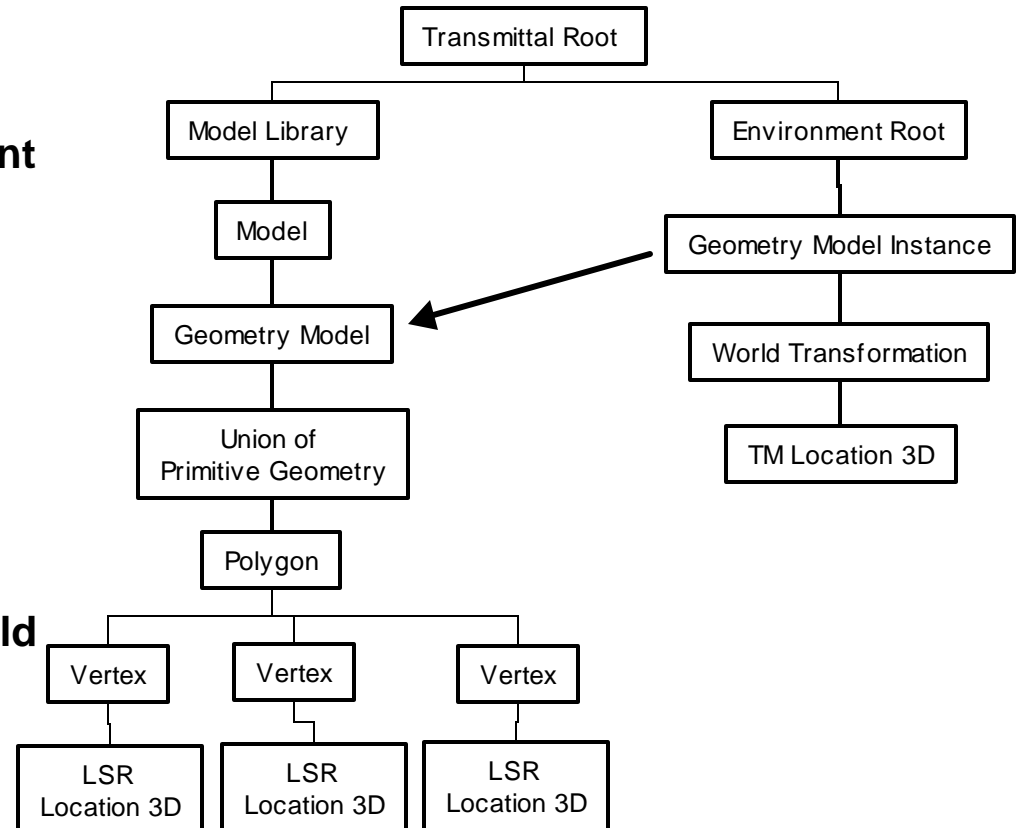
- **process_inheritance**
 - If set to SE_TRUE, returns inherited objects found higher in the aggregation tree
 - Removes interpretation
 - Correctly flows to the primitives
 - Simplifies the context issue





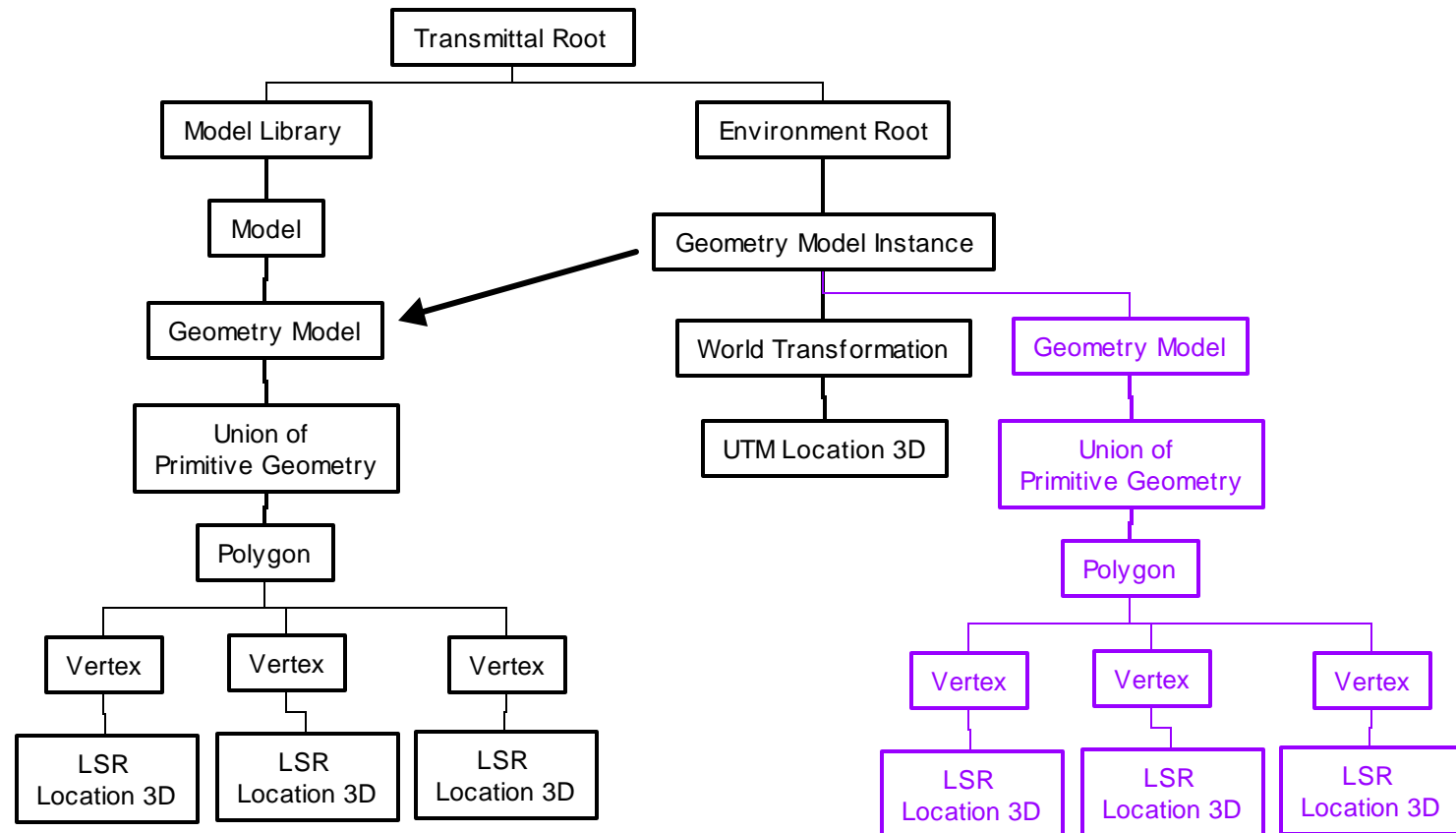
Making the API work for you [6 of 13]

- **Simplifying Model Instances**
- **Follow Model Instances**
 - Attaches the model as a component of the Model Instances
 - Instantiates all data that the <Model> contains
- **Transform Locations**
 - LSR Model locations instanced in the real world SRF
- **Evaluate Static Control Links**
 - Will automatically calculate the field values of <Model> data
- **Basic consumption strategies:**
 - Use API to instance models
 - Extract the <Model Library> before extracting <Environment Root>(s)
 - Application can transform models





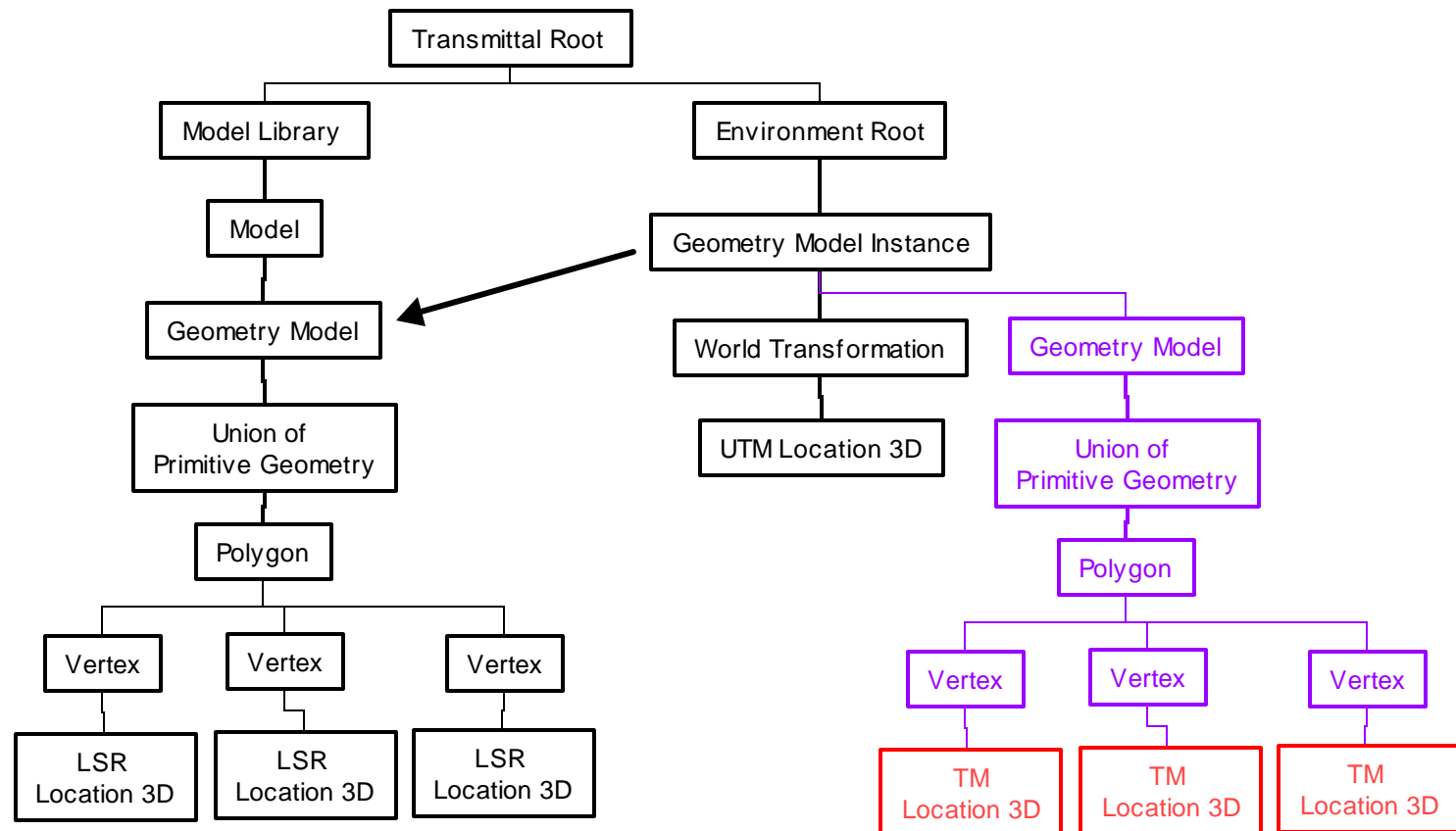
Making the API work for you [7 of 13]



- `follow_model_instances` set to `SE_TRUE`



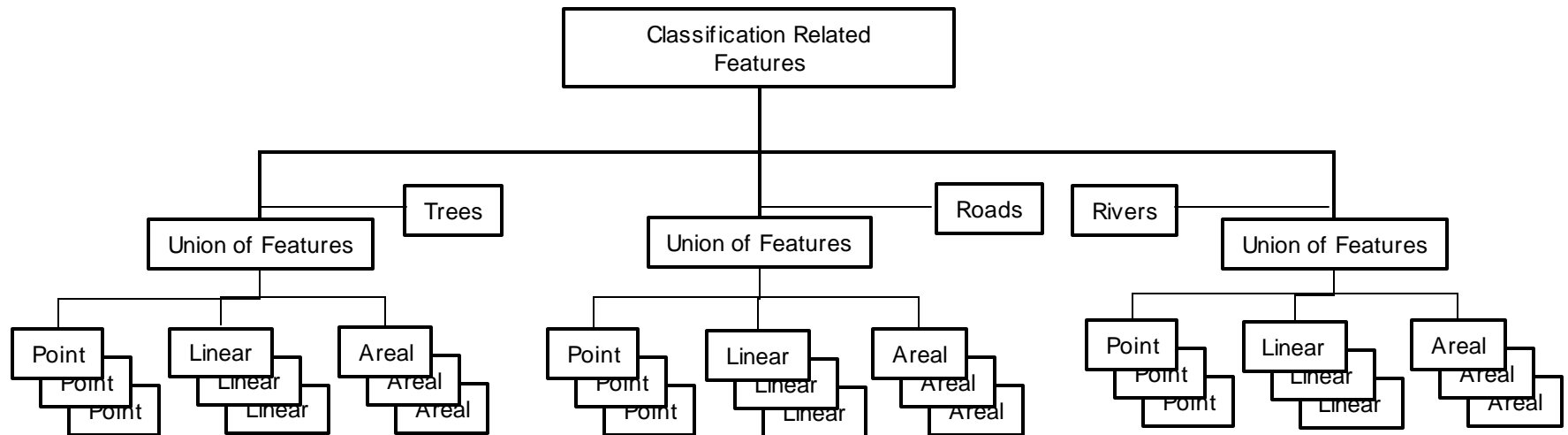
Making the API work for you [8 of 13]



- `follow_model_instances` set to `SE_TRUE`
- `transform_locations` set to `SE_TRUE`



Making the API work for you [9 of 13]

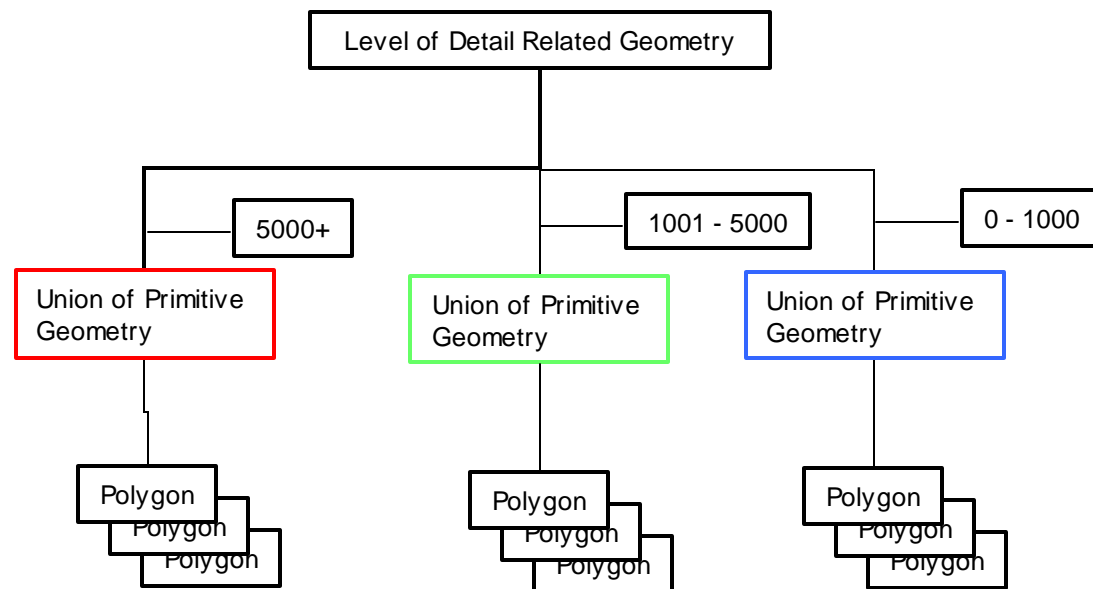


- **Hierarchy selection & filtering**
 - *Classification (ECC)*
 - *Attribute (EAC)*
 - *Spatial Index*
 - *Time*
 - *Level of Detail (LoD)*
 - *Quad Tree & Oct Tree*
 - *Separating Plane (G)*
- **Provides the capability to avoid paths in a hierarchy tree**



Making the API work for you [10 of 13]

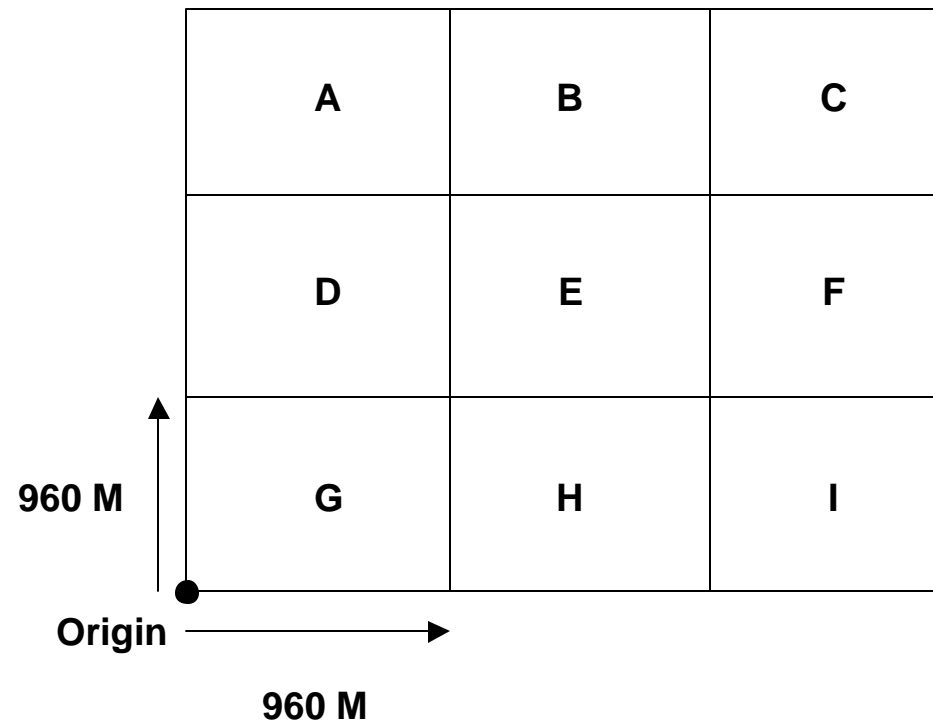
- **Hierarchy Ordering**
- **Will return components in descending or ascending order**
- **Example:**
 - *Ascending: Red, Green, Blue*
 - *Descending: Blue, Green, Blue*





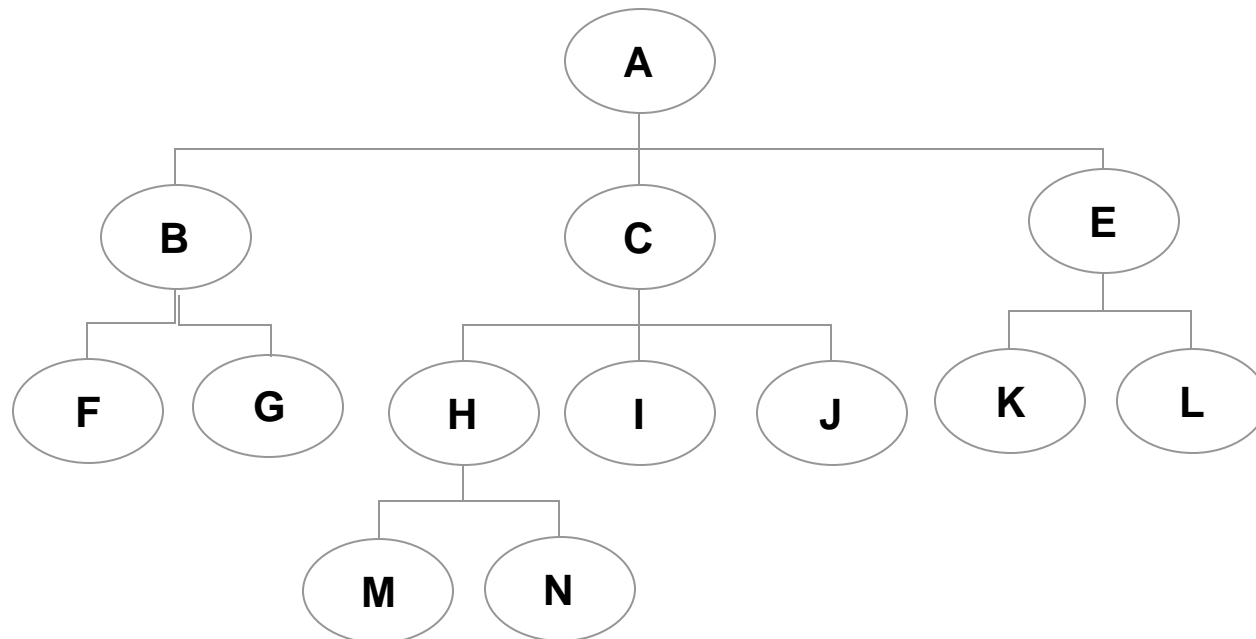
Making the API work for you [11 of 13]

- Hierarchy Ordering example: Spatially Indexed
- Ascending: G, H, I, D, E, F, A, B, C
- Descending: C, B, A, F, E, D, I, H, G





Making the API work for you [12 of 13]

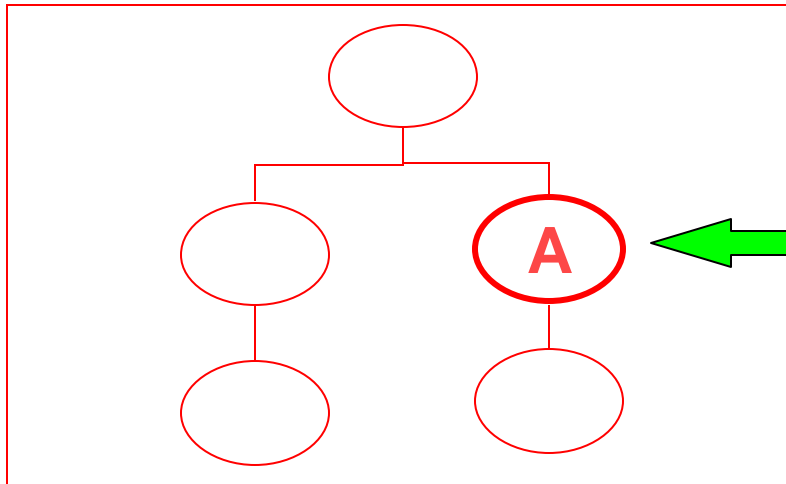


- **Depth first**
 - F,G,B,M, N, H, I, J, C, K, L,E
- **Breadth first**
 - B, C, E, F,G, H, I, J, K, L, M, N

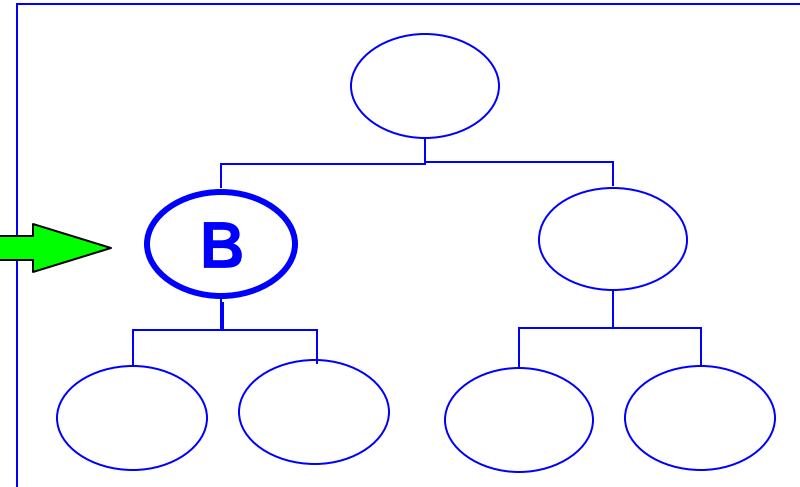


Making the API work for you [13 of 13]

Transmittal 1



Transmittal 2

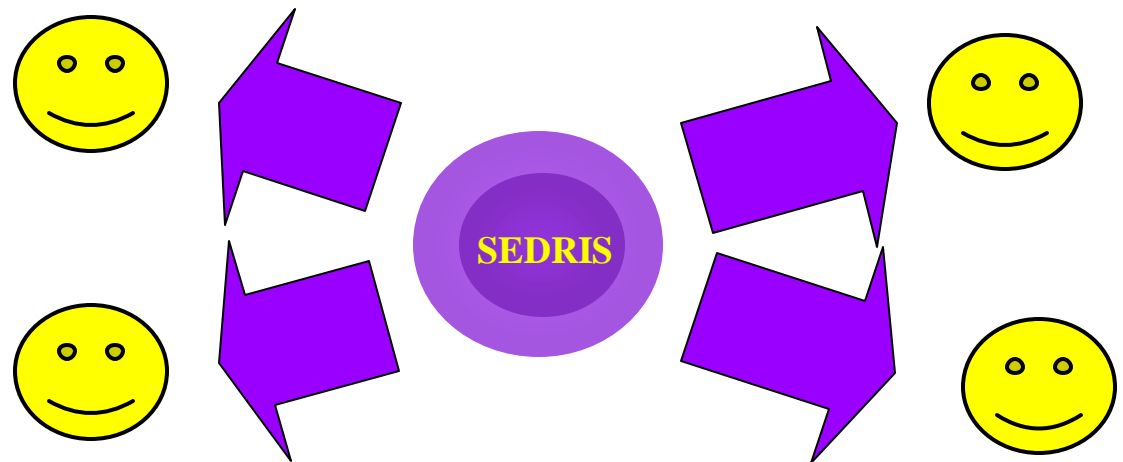


- **itr_traversal_order**
 - SE_ITR_BHVR_IGNORE
 - Won't see B
 - SE_ITR_BHVR_RESOLVE
 - Will return B
 - SE_ITR_BHVR_REPORT
 - Will say there is a B
- **Should always resolve**
 - API will do the work
 - Retrieve all the data



Step 6: Expand Consumption Base

- **Goal: Design for the future, implement on the pay-as-you-go plan**
- You can consume all possible SEDRIS Transmittals.
 - *But is that the best use of resources?*
 - *In other words, what would happen if everyone did that?*
 - *Duplicated Effort*
 - *Duplicated Code*





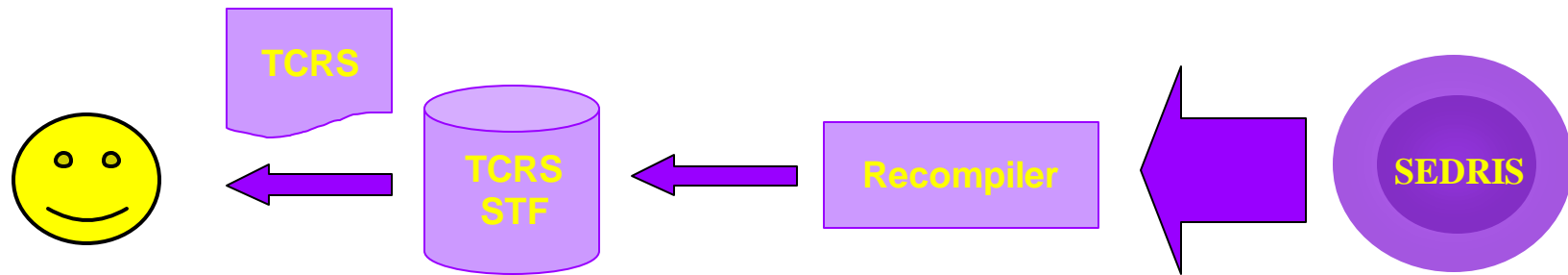
Step 6: Expand Consumption Base [2 of 3]

- **More efficient consumption**
- **Consumer focus**
 - Focus on SEDRI to native format
 - Not focused on consuming all SEDRI Transmittals
- **So, how does a consumer expand his consumption base?**





Step 6: Expand Consumption Base [3 of 3]



- **Recompiler**
 - Allows incremental development
 - Provides a clear mechanism for validation of transmittal
 - Allows for reuse of software components
 - Allows for development in SEDRIS components
 - Larger base of users
 - Larger code base
 - Larger market place



Case Study

- **STF 2 CTDB**
 - Published TCRS
 - Multiple passes of Transmittal
 - Allows users the capability to define which classification codes to use, specified at run time
 - Problems if encounters classification codes not defined at run time
 - Process libraries at one time and stores relevant information
 - Allows API to handle all inheritance issues



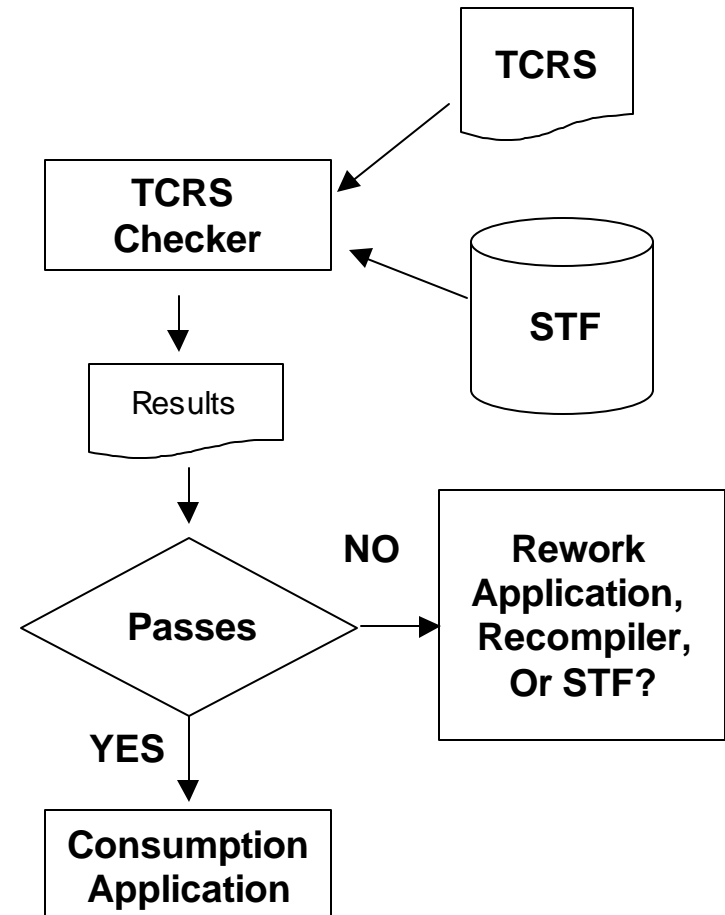
Case Study [2 of 2]

- **CCTT SAF**
 - Published CATT CDB TCRS
 - 2 Data providers: E&S & LMIS
 - Provided an application to re-organize STF into consumable organization
 - Must classify features with specific attribute code and classification codes
 - i.e., a tree shall have the following classification code with the following attribute codes
 - Strategy:
 - Limit consuming software to a closely defined STF organization
 - Provide application to create the closely defined STF organization from other STF organizations
 - Insulate the consumption software to deal with a subset of DRM organizations



Current Solution

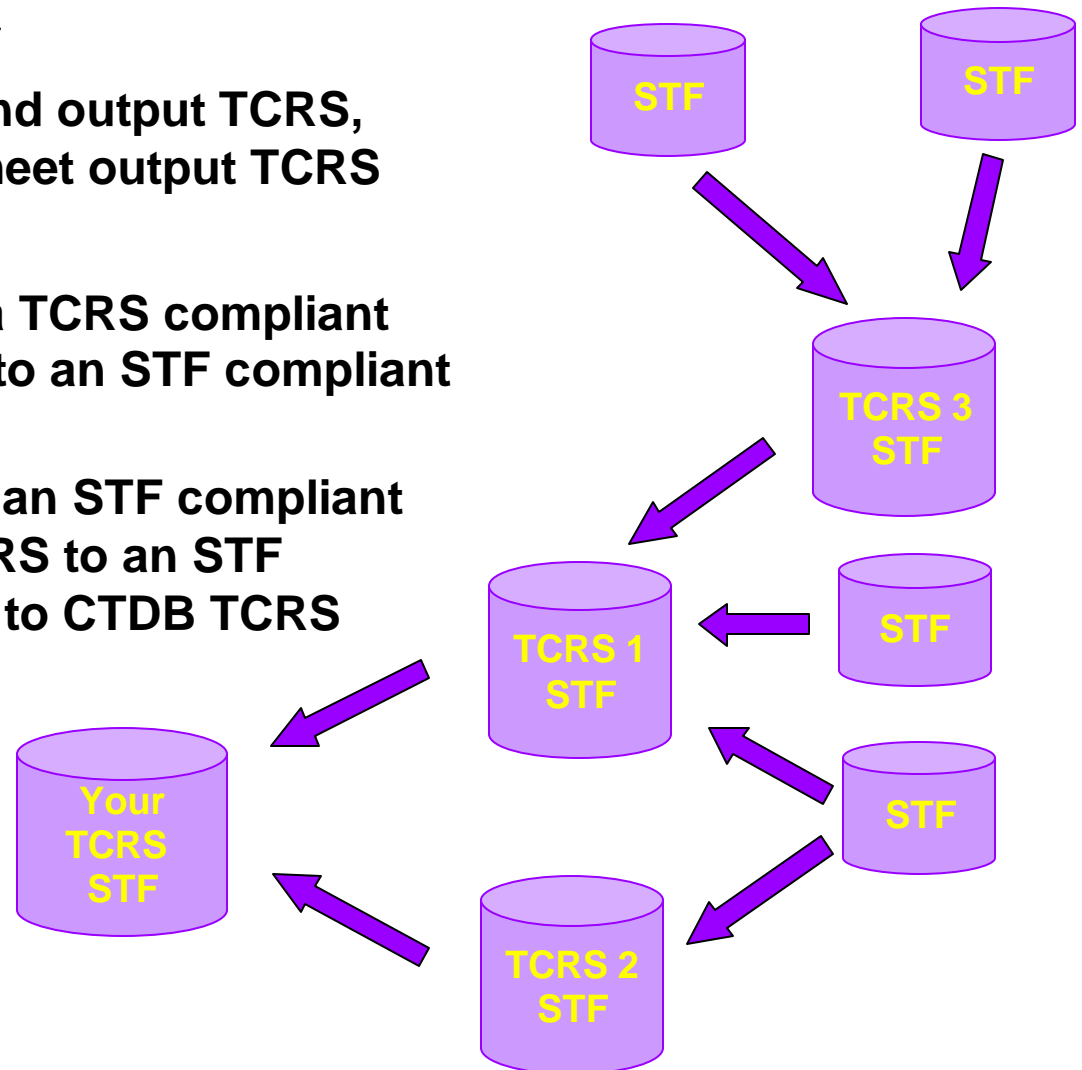
- **Bridging the gap**
 - Each step is a multiplier
- **TCRS XML encoding**
 - Allows for the clear specification of requirements
 - Allows for publishing requirements
- **TCRS_Checker**
 - Evaluates a transmittal's compliance with a TCRS
 - Provides failure instances
 - Gating point for consumption application
 - Provides consumer's flexibility on next step
 - Provides result's as to how close the STF comes to meeting your TCRS





Final Solution Path

- **Transmittal Transformer**
 - Using an input TCRS and output TCRS, transforms an STF to meet output TCRS
- **Incremental Step**
 - Recompilers that take a TCRS compliant STF and transform it into an STF compliant to another TCRS
 - Example: transforming an STF compliant with the CCTT CDB TCRS to an STF compliant with the STF to CTDB TCRS
- **Next Steps**
 - Graphical TCRS builders
 - Transmittal analyzer
 - Determine TCRS of a transmittal





Questions ?

- **Questions on Consuming?**
- **Documentation**
 - **SEDRIIS Technology Documentation Set**
 - **Part 4: *Technical Reference Set***
 - **Volume 15: *How to Consume SEDRIIS Transmittals***