Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Introduction (rev. 1)

February 14, 2008

AWP.TRG.SWCTR/TO6.ADE/CAVE00.01

Purpose of Course

- Early developer-level introduction to facilitate cooperative development
  - Trying to evolve project toward an Open Source core
  - Everything a part of the baseline and open to improvement
- Early focus on architecture and design patterns
  - Get the big picture right, before moving into specific capabilities
  - Widen exposure to get more creative input
- Workstations with full installation of ADE 1.0
  - Source with Eclipse IDE
  - Server Side Run environment
  - CAVE visualization
  - Javadocs and other documentation

Training Prerequisites

- Reading materials
  - *Software Product Improvement Plan*
- Software
  - Pure Java
    - http://java.sun.com/docs/books/tutorial
  - CAVE: ECLIPSE IDE Framework & Plug-Ins
    - http://www.eclipse.org
    - Eclipse RCP
  - EDEX: Introductory level of Spring and Mule ESB
    - http://mule.codehaus.org
  - All: Introductory level of ANT
  - All: Introductory level of XML

Course Objectives

Module 1: Architecture

- Understand the overall System Architecture

Module 2: Installation, Build, and Regression Test

- Successfully install ADE
- Have ADE ready for running and development
- Successfully do a “Clean Build and Deploy”
- Successfully verify system installation by running a standard regression test through a regression test GUI (e.g., Tomcat)
- Learn how to use “Debug” to step through code running in services

Course Objectives (cont'd)

Module 3: MicroEngine Scripting

- Understand how to create tasks and scripts for the MicroEngine (uEngine)

Module 4: Data Type Plug-In

- Learn why the Plug-In Pattern was chosen
- Understand the architectural pattern of a Data Type Plug-In
- Write a new Plug-In for a new data type
- Put a MicroEngine task into a Data Type Plug-In

Course Objectives (cont'd)

Module 5: Service Oriented Architecture (SOA)

- Understand the architectural pattern of an SOA service
- Understand how services are written
- Understand how services are integrated into the system
- Understand how to monitor and test an SOA service

Module 6: CAVE-Underlying Framework and Rendering

- General introduction to CAVE
- Understand how CAVE renders geospatial, vector, and x-y data

Course Objectives (cont'd)

Module 7: CAVE-User Interface

- CAVE baseline orientation
- Add functionality by modifying plugin.xml
- Add a new menu item and custom resource

Module 8: CAVE Visualization Plug-Ins

- Understand the mechanisms to extend CAVE
- Write a new Plug-In to extend CAVE functionality

Module 9: Installation/Deployment

- Install the EDEX services and CAVE application to a supported platform

Course Objectives (cont'd)

Module 10: CAVE Menu Creation

- Describe the changes to Menu Architecture in made in TO6
- Provide an example of creating a new menu in CAVE

Module 11: Localization

- Introduce the Localization concepts in ADE 1.0
- Describe the new Localization process

Course Objectives (cont'd)

Module 12: TO8 ADE 1.0 Developer Updates

- Describe revised Ingest Data Flow
- Describe revised Data Access Layer implementation
- Describe revised Database definition pattern
- Review modifications to the Plug-in creation utility
- Review Installer modifications for TO8

BREAK


Innovation • Partnership • Responsibility
The Raytheon Mark's Team

Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 1: Architecture (rev. 1)

February 14, 2008

AWP.TRG.SWCTR/TO6.ADE/CAVE01.01

Introduction

- **Early Decisions and Concerns That Drove Architecture**
  - How to deal with changing ConOps
  - How to add new data types quickly
  - Adding new science
  - Scale to increasing data rates
  - Lower sustainment costs

- **Some Core Principles**
  - Minimize coupling
  - Increase cohesion
  - Minimize size of code base
  - Maximize simplicity
  - Pull style data flow

Architecture History Leading to SOA

- Evolved out of the message-based approaches from 20 or 30 years ago for high-performance systems
- In the 1990s: Systems built on message busses like DecMessageQ, Tibco, MQseries
- Evolved into J2EE and JMS (Java Messaging System) – currently used by many systems
- SOA: Somewhat of a rebellion against the unnecessary complexity of J2EE for some domains
- Decision to take the next step by decoupling the physical service from the communication mechanisms through the use of Enterprise Service Bus (ESB)

Core Decisions – Use ESB and Container-Based Processing

- ESB with Execution Container
  - Startup, shutdown, communication, multi-threading
- MULE + SPRING Execution Container
  - Dependency injection (minimizes coupling)
  - Example: Look at PersistSrv.java

Outbound Endpoint Hit on return statement in service

```
<mule-configuration version="1.0">
        <model name="awips" type="seda">
        <mule-descriptor name="Awips.Edex.Service.PersistSrv"
                singleton="false" inboundEndpoint="vm://persistVMQueue"
                outboundEndpoint="vm://indexVMQueue"
                implementation="com.raytheon.edex.services.PersistSrv">
                <threading-profile maxThreadsActive="5" maxThreadsIdle="5" />
                <properties>
                        <container-property name="dataLayer" reference="DataLayer"
                                required="true" />
                </properties>

        </mule-descriptor>
        </model>
</mule-configuration>
```

Core Decisions (cont'd)

- JAVA as primary programming language
  - Makes plug-ins possible, interfaces, classloading
  - Traction in commercial and Open Source programming
  - Performance comparable to other approaches
  - Programmer productivity improvement
- XML as primary text format for messages and configuration
  - Self-describing, platform independent, standard parsers
- Plug-In: The mechanism for extending the system
  - Can be Hot deployable, or system cycled to pick up new plug-ins

System Concept: AWIPS Architecture Environmental Intelligence Framework

- Requirements Vision drives Architecture
  - Focus on “ilities” drives new AWIPS Architecture
  - Features and capabilities get generalized into reusable patterns
  - Customer TIMs give priority to capabilities
- Architecture Framework Vision
  - Create a new, low-cost framework for hosting a full range of environmental services, including thick client visualization
  - Scale down Framework to a small laptop and up to clusters of enterprise servers without software change
  - Base Framework on highly reusable design patterns that
    - Maximize reuse
    - Have datatype independence and fast adaptability
  - Open Source leveraged to maximize reuse

AWIPS Architecture Definitions

Term	Definition
ADE	AWIPS Development Environment; source code to Execution Framework Enterprise Development Kit, including tools
SOA, End Points, I/O Routing, Transforms	Service Oriented Architecture where system capability is available at stateless endpoints
Canonical XML	Well-formed XML that follows high-level rules
Patterns	Implements a design solution that solves a problem that occurs many times
Technical Reference Architecture	A physical Software Execution Framework
JMS, JMX	Java Messaging System (API), Java Management Extensions
CAVE	Common AWIPS Visualization Environment
SEDA	Serial Event Driven Architecture

AWIPS Architecture Concept – Architecture Framework Implementation

Raytheon

Concept

- Framework Implementation: Integrated several “best of breed” Open Source projects with a set of advanced enterprise patterns to create a highly extendable framework
  - Patterns implemented in pure Java (reuse example: ProductSrv and AutoBldSrv use uEngine pattern)
  - Open Source primary source of reuse
    - 15 major Open Source projects integrated
    - Version controlled with CM baseline, libraries part of run environment
    - Leverages Internet community for core infrastructure
    - Standards compliant, rapid evolution
    - Free, large body of public expertise
    - Open Source libraries controlled by putting them in the CM Compile Library and deploying them to the runtime environment
  - Packaged together in the ADE, which contains everything from the Source Code repository to the execution environment, including operator Clients

AWIPS ADE Open Source Projects: Integrated Open Source Projects

Raytheon

Concept

ANT	Build scripting	ADE build system
Mule + Spring	Enterprise Service Bus + Container	ADE Run Environment
ActiveMQ	Java Messaging System Broker	ADE JMS Broker
Jibx	XML to Object Serialization	ADE Canonical XML Message
GeoTools + JTS	GIS capabilities	ADE GIS primitives
Tomcat	Web Server	ADE Test Client Server
Jython	Python Scripting Engine	ADE uEngine Python Script Engine
Baltik	Scalable Vector Graphics Tools	ADE SVG tools
Rhino	Java implementation of Javascript	ADE uEngine Script Languages
Ehcache	Event Driven Clusterable Cache	ADE Cache Framework
Log4j	Java Logging API	ADE Log manager
Jogl	Java API to OpenGL	ADE CAVE rendering interface
Eclipse RCP	GUI plug-in based framework	ADE CAVE framework
Eclipse IDE	Java Integrated Development	ADE development environment
MC4J Console	JMX Management Console	ADE remote management console

SOA Framework Concept
Extensible Architecture – Minimal Coupling

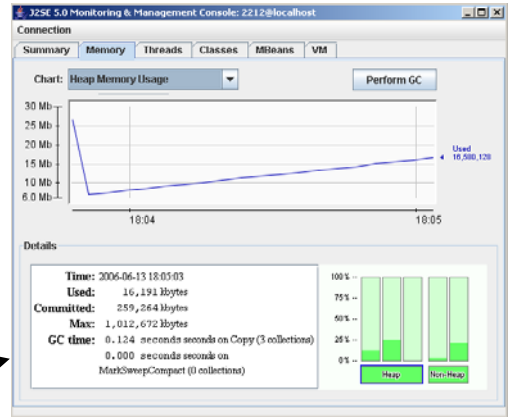
Raytheon

Features

Core Base of Services
- MicroEngine
- Plug-In Framework
- Extensible XML model
- Core SOA Services

<Technical Reference Architecture>
Services

+Ingest Data	+Request
+Index Data	+Store Data
+Adapt	+Auto Build
+Subscribe	+Collaborate



Extend to a Specific Domain
- Plug-in specific libraries
- Plug-in data types, transforms

Satellite Feed 4.5Mps

Local Data

<Technical Reference Architecture>
SOA System Services (EDEX)

+domain libraries

+mapping libraries

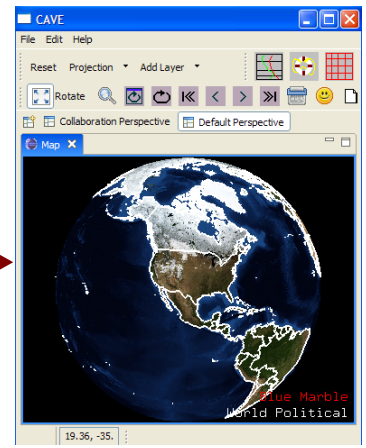
Meta Data

Data

Data Types

Transforms

Scripts



Service Interface To Data
- Clean separation between data and visualization
- Canonical XML data model
- Scriptable Interface

<Visualization Framework>
CAVE

2D/3D GIS	Drawing
Raster Render	Collaboration
Vector Render	XY Render
Auto Animate	Localization

Vis Plugin 1

Vis Plugin 2

Vis Plugin n

New in TO6

Architecture Features: Execution Container & Data

- Enterprise Service Bus (ESB)
  - Combined approach to integration: provides plumbing for highly distributed, loosely coupled services
  - Dependency Injection Container: Minimizes service and component coupling, makes for more flexible services
  - Messaging, Web Services, Data Transformation, Routing
  - Process flow and service invocation (can span entire bus)
  - Provides clear separation between business and control logic

- Data Persistence
  - Use of a simple retrieval-oriented metadata model that is keyed to high-performance HDF5 file persistence
  - Considered a fully normalized RDBMS model but adds only limited value for transient weather data; the complexity not considered worth it

Architecture Features – Geospatial
Spatial-Enabled PostgreSQL and GeoTools with JTS

Raytheon

Features

- Geospatial enabling data
  - Chosen approach: Create static spatial tables in PostgreSQL
    - PostGIS extension: Free, simple, high performance
  - Visualization: Use ESRI Shape Files as standard vector format
    - Enables GIS analysis of data, also renders SVG using “batik”
    - Renders GeoTiff using Tiff tags
  - Input / Output
    - All ingested data spatially indexed and can be spatially queried
    - Can create Shape and GeoTiff output

radar_spatial

gid [PK] serial	rda_id character varying(4)	wfo_id character varying(4)	eqp_elv numeric	name character varying(36)	lon double	lat double	elevmeter double pre	the_geom geometry
1	TADW	LWX	346.1500	Andrews AFB, MD	-76.845	38.695	105.50652	0101000000A

spatial_grids

grid [PK] integer	description character varyin	dy double pre	crs character varying(2047)	coverage geometry
87	U.S. Area; used in I	68.153	PROJCS["Polar Stereographi	0103000000010

spatial_obs_stations

gid [PK] serial	icao character var	wmoregion integer	elevation integer	upperairelev	rbsnindicator character var	stationgeom geometry	upperairgeon geometry
1	AGGH	5	8	9	P	01010000009A9	

spatial_satellite

id [PK] ch	description character varying(255)	crs character varying(20	coverage geometry
TICF01	Multi-Satellite Composite	PROJCS["Polar Stereogra	0103000000010

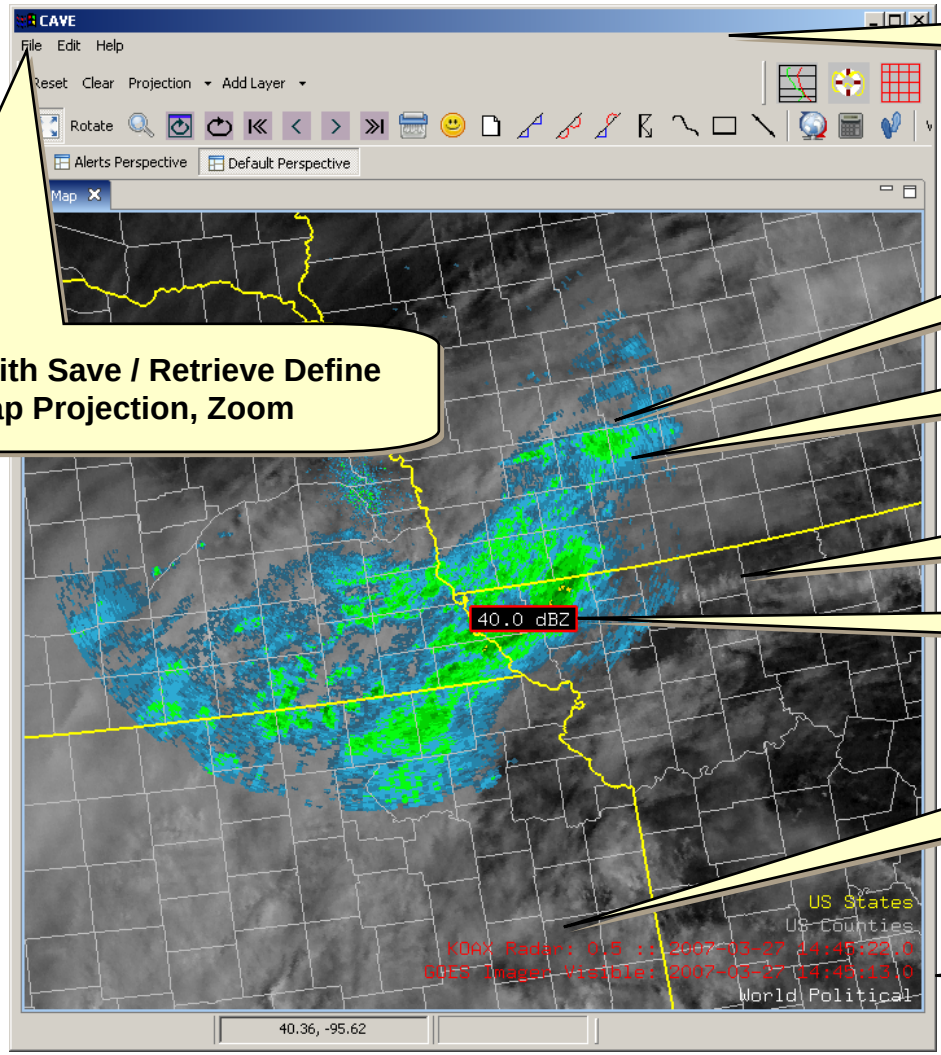
Architecture Features: Visualization

- Approach
  - Develop a Common AWIPS Visualization Environment (CAVE)
    - Supports the fixed scales and detailed interactions of D2D
    - Supports large data sets and analysis capability of N-AWIPS
    - Supports GIS visualization and analysis natively
    - Supports collaboration and remote Client operations
  - Build on the Eclipse Rich Client Platform: Full-featured framework with an extensive widget set, extendable through plug-ins, high performance, Open Source
  - CAVE: A set of Eclipse plug-ins installed in the Eclipse RCP
  - Extensive support community, large public repository of plug-ins, several graphics-intense applications being developed in it

ADE CAVE Visualization: Service End Point Enables Gaming Style Data Interactions

Raytheon

Features



Eclipse RCP 3.2 – Plug In Extendable Plug In for Warn Generation Added

T05 Radar Rendering Uses Dynamic raster tiling

Bundles with Save / Retrieve Define Layers, Map Projection, Zoom

GPU Shader Language Rendering Controls (Color, …), Animation

New Quad tiling of large raster Sets leverages HDF5 chunking

Active Raster Data Interrogation

Dynamic map reprojection using GeoTools Transforms& GPU Warping

All Tilts Keyboard Controls

Architecture Features: Languages, Interprocess Communication

Raytheon

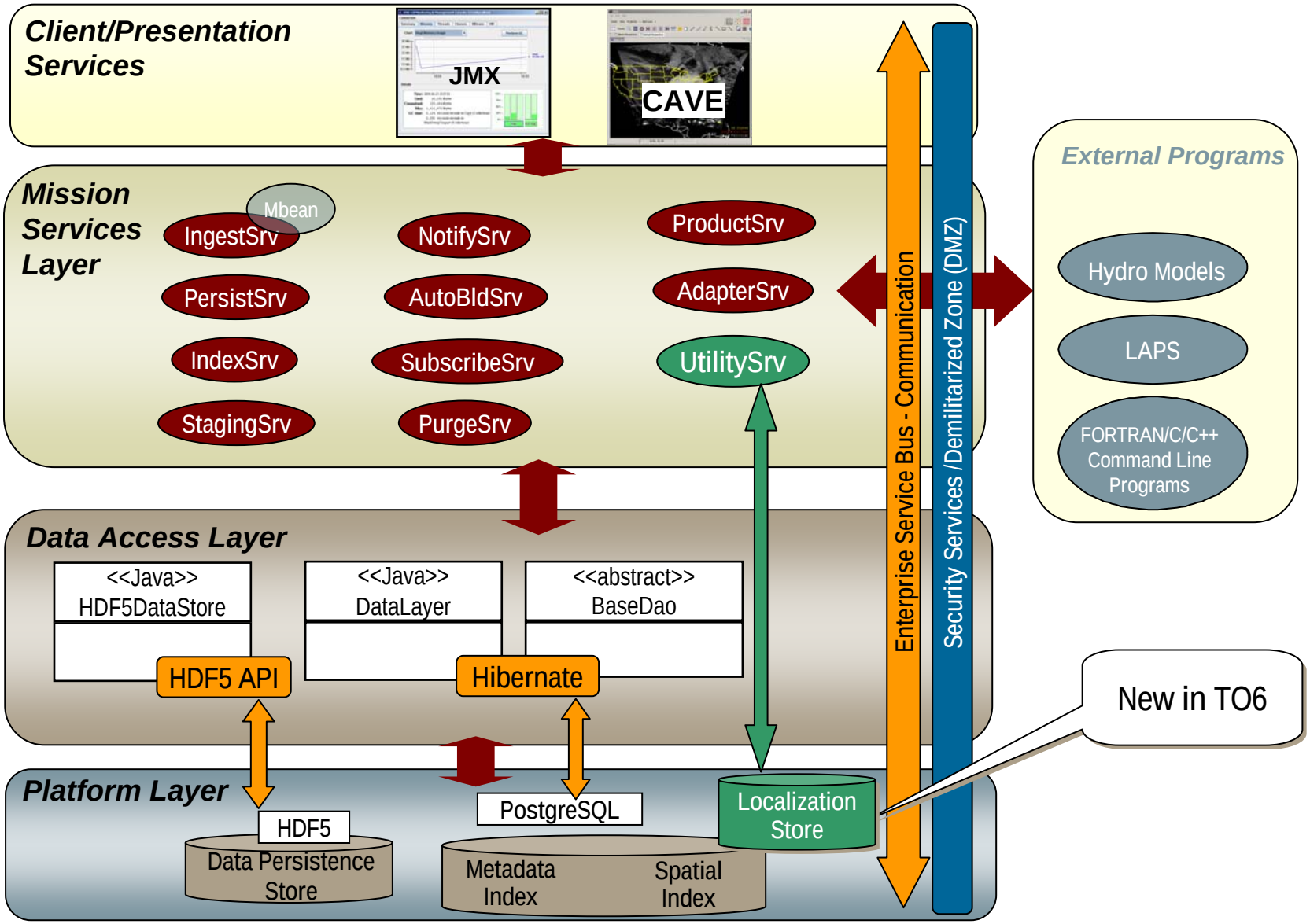
Features

- Java the Primary Programming Language
  - Extensive Open Source support, high programmer productivity, high reuse, performance parity with traditional languages, university teaching language
  - Enables platform independence
- Rhino (JavaScript) for scripting
  - Extensible with Java classes, large base of customer scripts and expertise, clean OO approach to scripting
- JMS primary for interprocess communication
  - Enables SEDA processing, increases reliability through queue persistence, enables subscription / notification through topics
  - Enables asynchronous communication for performance
- HTTP, FTP, JMS, E-Mail for WAN Communication
  - Firewall compatibility, enables CAVE to act like a thin client, can transparently switch between JMS and HTTP without application changes

Conceptual Architecture: Logical Layered Viewpoint

Raytheon

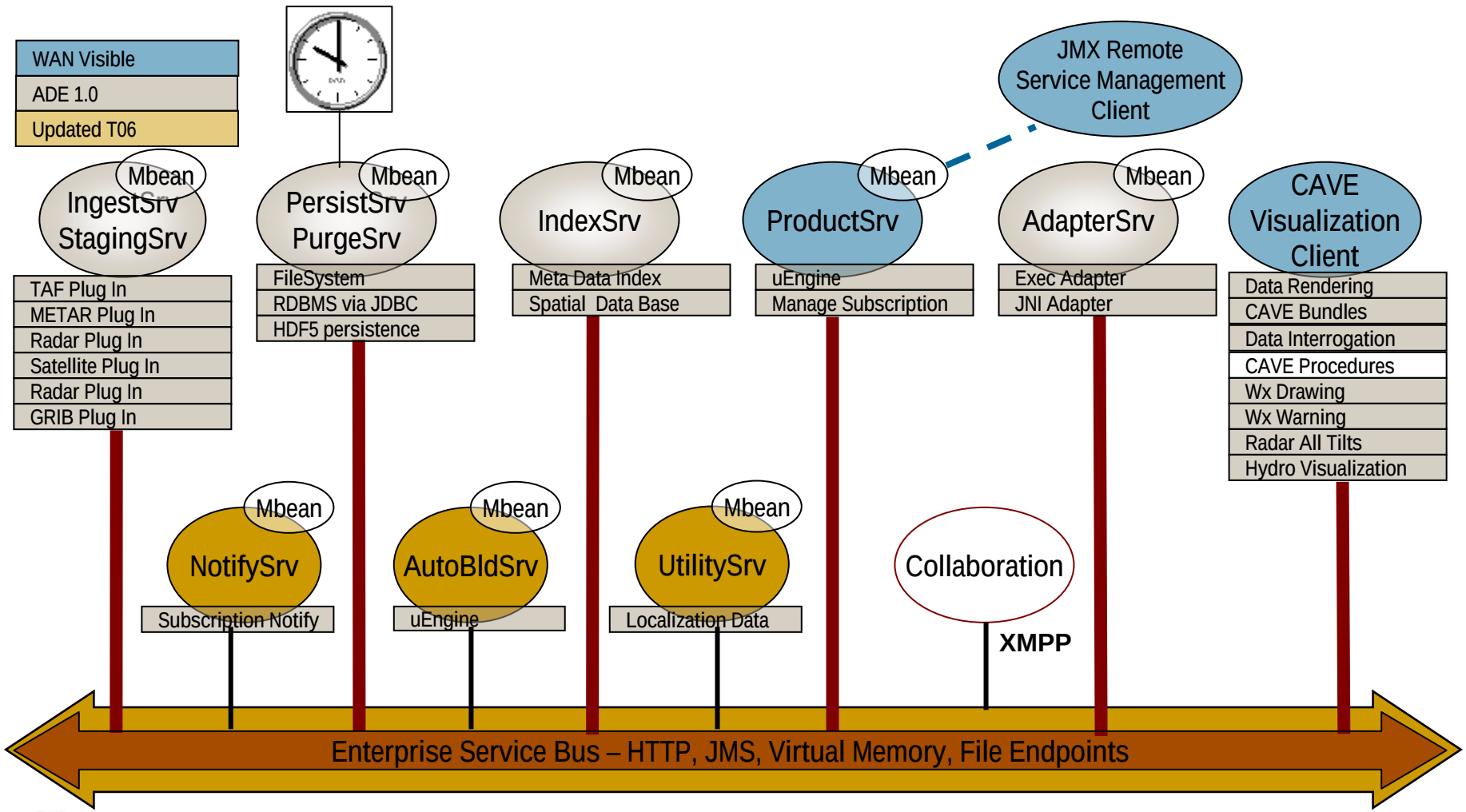
Features



AWIPS II ADE High-Level System Services SOA Services Running in an ESB Container

Raytheon

Features



Services Independent of End Points

Service Descriptions

Service	Description
IngestSrv	Listens on an endpoint for new data and transforms the data into a message
PersistSrv	Writes ingested data to a persistent store file system or RDBMS
PurgeSrv	Runs periodically to maintain the metadata and the persistent stores
IndexSrv	Indexes the metadata extracted from the ingested data into a store that facilities data searches and retrievals
ProductSrv	Listens on an endpoint for external product requests and fulfills requests with a response message. Typically receives “Action” scripts that describe how to transform raw data into a visualization product
NotifySrv	Broadcasts a product from a subscription fulfillment. Also sends out alerts based on ingested data
AutoBldSrv	Receives requests to build products that are under subscription. Triggered by data arrival and/or time
ColaborateSrv	Provides common point for serving out products shared by several clients
AdapterSrv	Enables legacy command-line programs to be run as a standard service

ADE Implemented Design Patterns
Patterns Enable AWIPS "ilities"

Raytheon

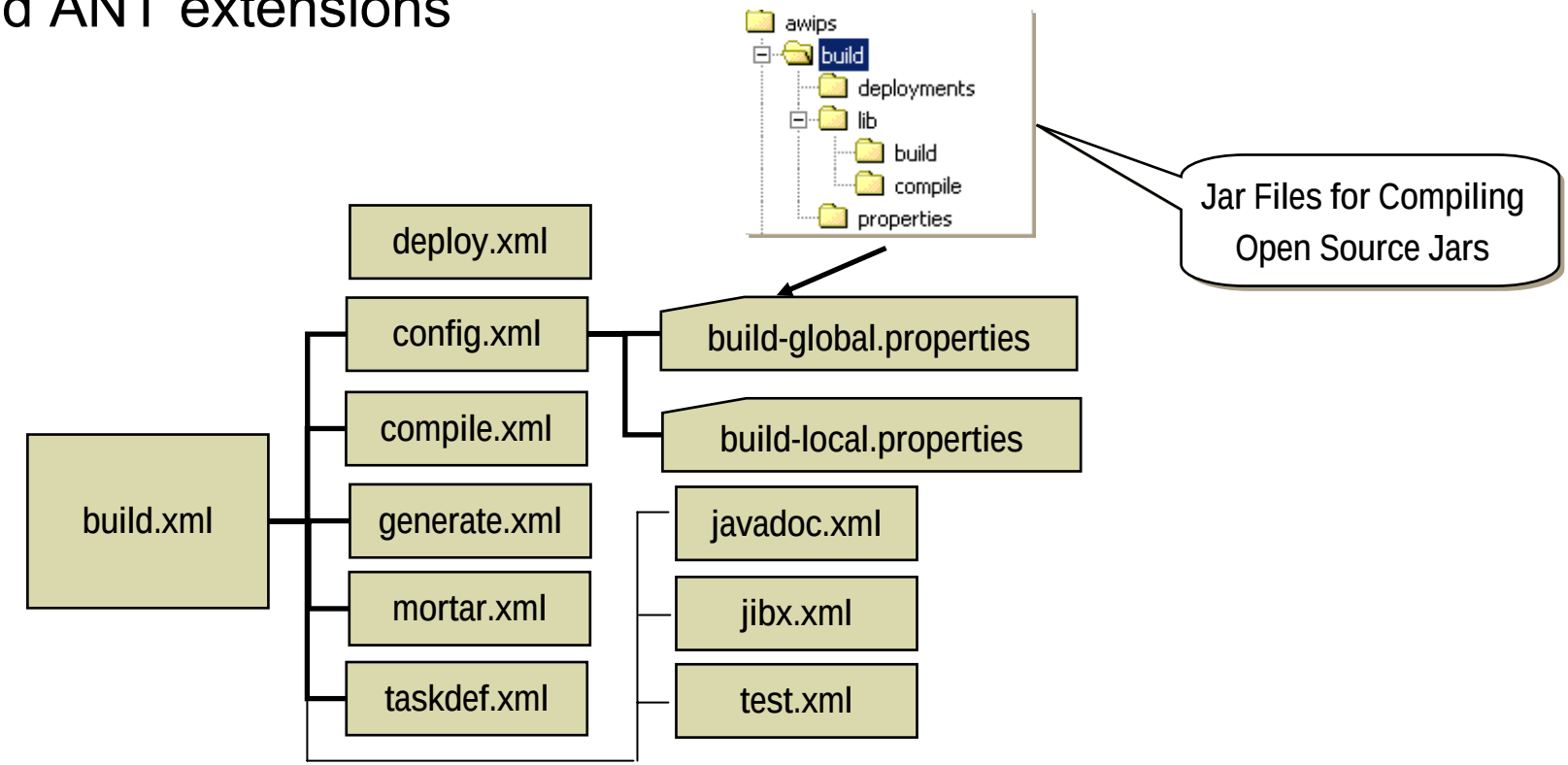
Patterns

Focus on Patterns That Maximize Reuse Across System Functions

CM/Build/Deploy Pattern	Use Open Source tools to standardize build and enforce standards for components
SOA Service Pattern	Simplifies Service interactions with application containers
Canonical XML Service Interface	Standardizes the request / response interface to SOA services
Component Model	Standard pattern for injecting new components
uEngine Task Execution Pattern	Enables system flexibility through re-use of small units of execution
Geo Spatial Pattern	Enables building, displaying, analysis, and querying for data
Datatype Plug-in Pattern	Enables system adaptability to new data and transforms
Legacy Adapter Pattern	Enables system evolution by allowing legacy processes to run in an SOA
Data Notification / Subscription	Enables data driven processing and display
Common AWIPS Visualization Environment (CAVE)	Consolidates disparate display mechanisms into one platform independent whole

Software CM/Build/Deploy Pattern Design Pattern

- **Build Vision:** Create a simple layered build system that manages component coupling and supports partial deployment
- **Build Implementation:** Implemented in ANT as a series of macros and ANT extensions

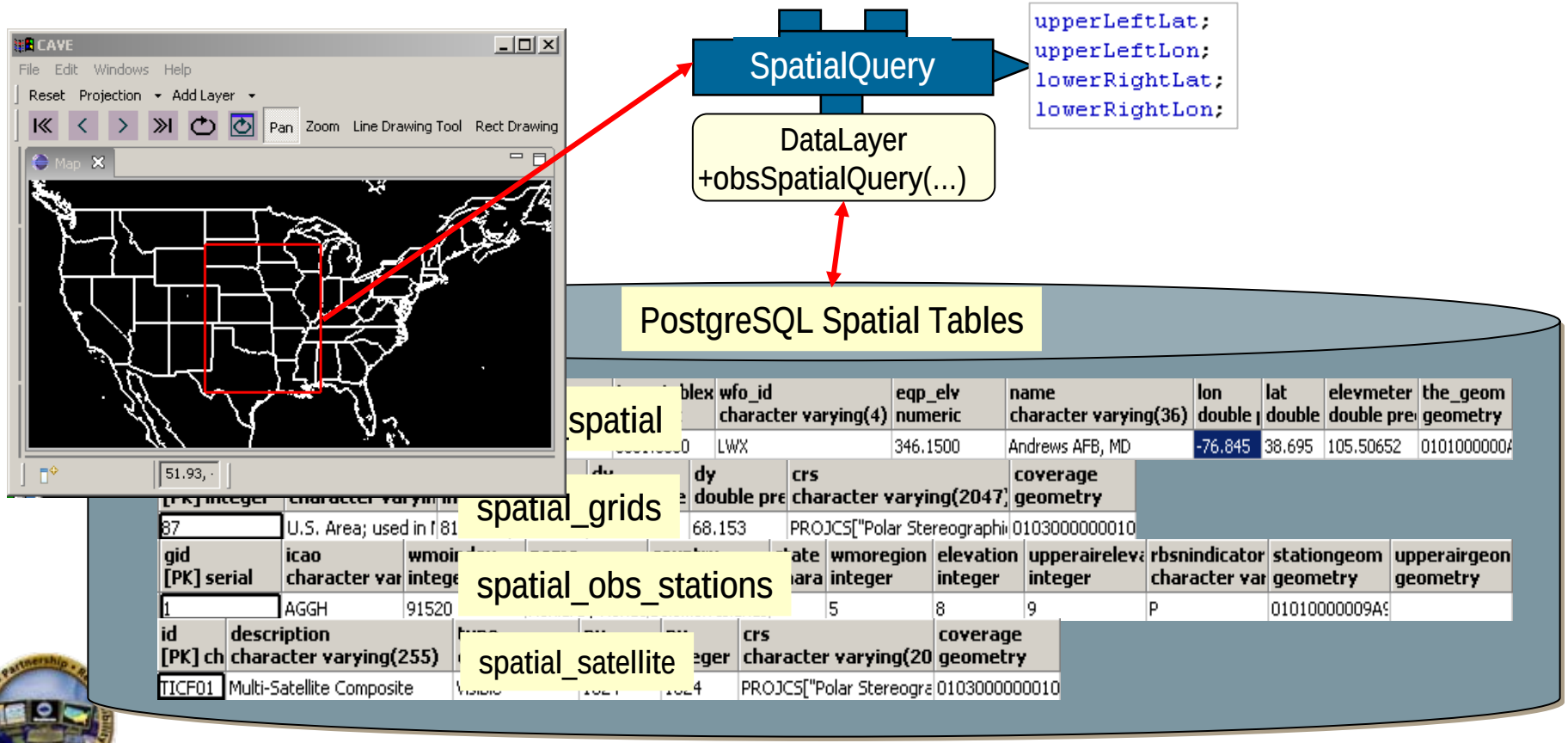


Geospatial Pattern
Basic GIS Ingest, Indexing, Output, and Analysis

Raytheon

Spatial Pattern

- **Geospatial Vision:** All ingested data indexed by spatial index making Spatial Query and Analysis available to the Visualization Operator or SOA service



Geospatial Pattern
Coordinate Reference System (CRS)

Raytheon

Spatial Pattern

- Spatial Descriptor: completely defines a grid area
  - GeneralEnvelope: Geo Tools Concept
  - GridGeometry2D: Geo Tools
  - Coordinate Reference System

```
<crs percentOfEarth="1.0">GEOGCS["WGS 84", ^M
 DATUM["WGS_1984", ^M
   SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]],
   AUTHORITY["EPSG","6326"]], ^M
 PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]], ^M
 UNIT["degree", 0.017453292519943295], ^M
 AXIS["Lon", EAST], ^M
 AXIS["Lat", NORTH], ^M
 AUTHORITY["EPSG","4326"]]</crs>
<gridSize gridX="20000" gridY="10000"/>
<gridEnvelope
        minX="-180.0"
        maxX="180.0"
        minY="-90.0"
        maxY="90.0"/>
<pixelExtent
        minX="3825.6462531248258"
        maxX="5115.6928438385385"
        minY="2823.323080879442"
        maxY="3673.164236066635"/>
```

Coordinate Reference System (CRS) and Grid definition from a bundle.xml file in CAVE

BREAK


Innovation • Partnership • Responsibility
The Raytheon Team's Team

AWIPS Data Models

Data Model	Description
Service Interfaces Data Model	Canonical XML model, message format for external interfaces to SOA services
Metadata Model	Key fields and their definitions for ingested transient data
Data Object Model	Java OO model for internal data representations ▪ Data in Object Model also has XML representation through JIBX
Data Persistence Model	For transient data storage
Static Data Model	For data that seldom changes ▪ Data in Object Model also has XML representation through JIBX ▪ Map Scale Areas ▪ Station Data ▪ Map overlays (ERSI shape files stored on disk)

Data Model Introduction
Canonical XML SOA Interfaces Excluded

Raytheon

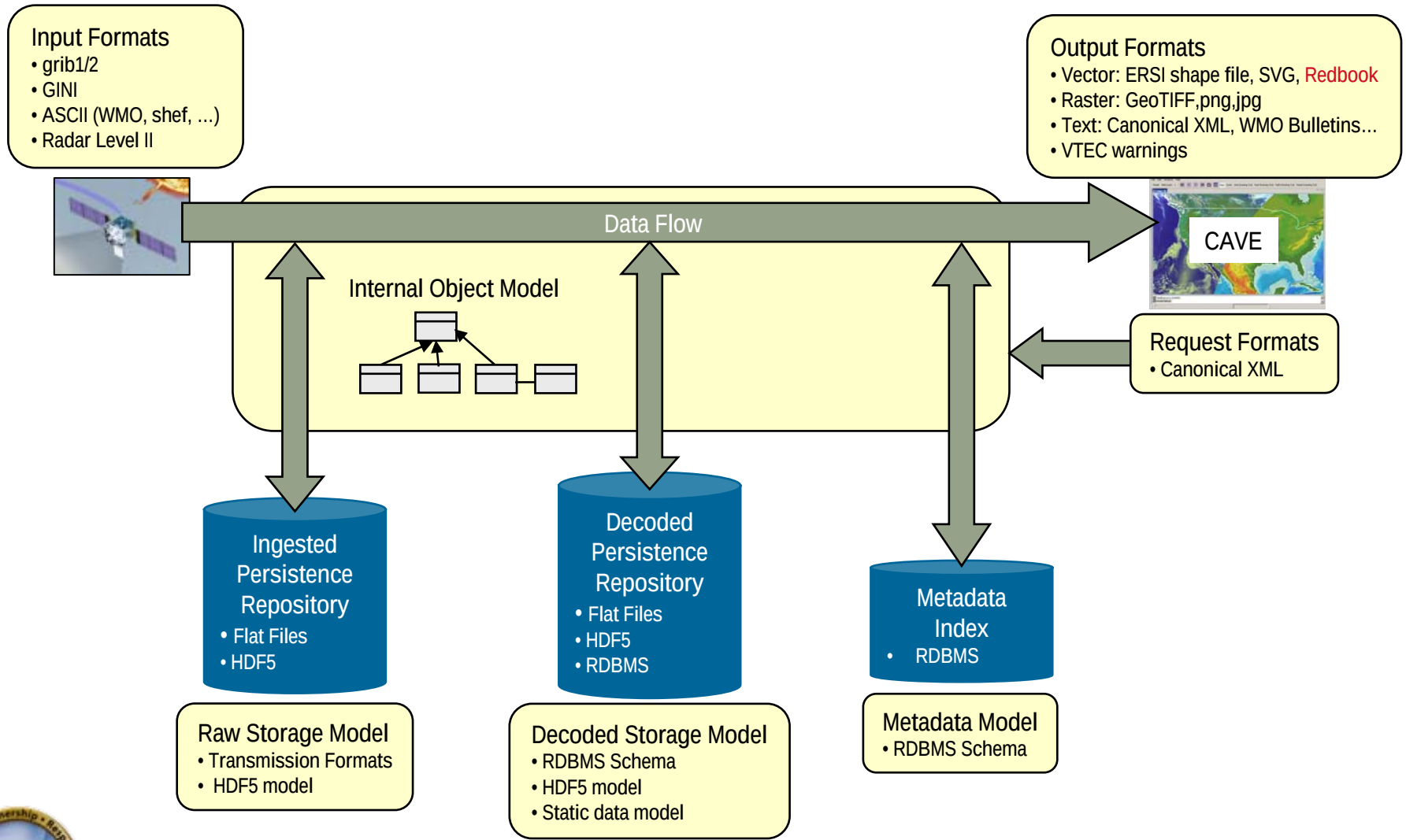
Data Model

- Data Access Layer implementation using Hibernate
- Data Access Object (DAO) concept leverages Hibernate
- Data Persistence through HDF5. Why?
  - High-performance gaming-level interactions supported
  - Chunking of data records supports visualization tiling
  - Flexible retrieval supports 4D rendering
  - Streaming compression
- Metadata implemented in PostgreSQL through Inheritance
  - Defined only in plug-ins, drives Data URI
- Base Object Model extended in plug-ins
- Data URI concept ties everything together
- Purging concept of circularly repository structures

Conceptual Data Model Design
I/O Formats Follows Existing Standards

Raytheon

Data Model

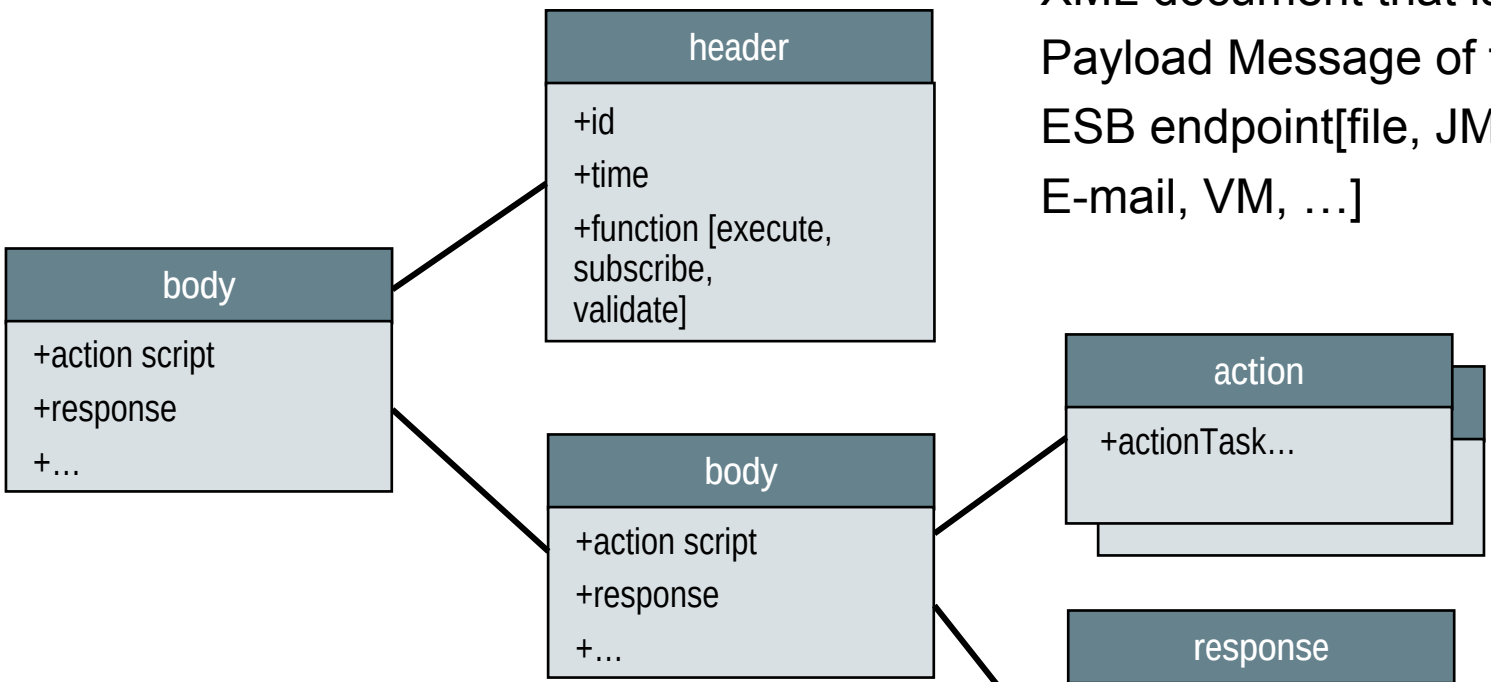


AWIPS Canonical XML – Top Level Structure (End Point Independent)

Raytheon

Data Model

XML document that is the Payload Message of the ESB endpoint[file, JMS, http, E-mail, VM, …]



Well-formed XML that can be parsed by DOM, SAX, common-digester, …: dynamically extendable

Data Access Layer API
Hibernate Leading Object to Relational Approach

Raytheon

Data Model

- Solves fundamental problem of impedance mismatch
- Maps between Object Model and Relational Data Model
- Provides object-based query facilities
- Improves performance over JDBC; designed for clustering
- Reduces code count; improves productivity
- Built-in support in SPRING

Hibernate Enables Metadata Performance and Adaptability

Hibernate XML Object/Relational Mapping Defined in SOA Plug-In: Enables Adaptability

Raytheon

Data Model

Satellite Plug In

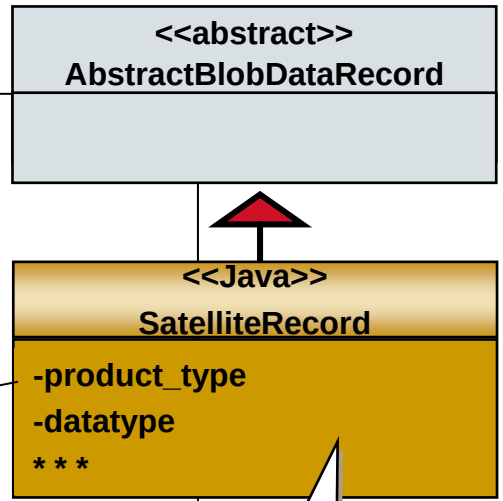
satellite.hbm.xml

SatelliteRecord.java

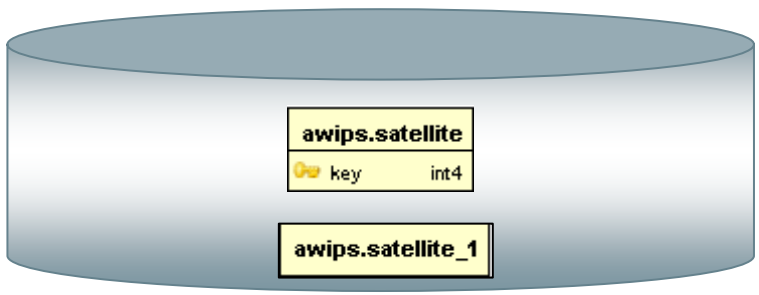
```
<hibernate-mapping default-lazy="false" schema="awips">

        <class name="com.raytheon.edex.satellite.SatelliteRecord"
                table="satellite">

                <id name="id" column="key" type="java.lang.Integer">
                        <generator class="native" />
                </id>

                <property name="dataURI" column="datauri"
                        type="java.lang.String"/>

                <property name="product_type" column="product_type"
                        type="java.lang.String" />
```

mapping

Data Object Model



Plug-In Defined Object to Relational Mapping by Hibernate XML



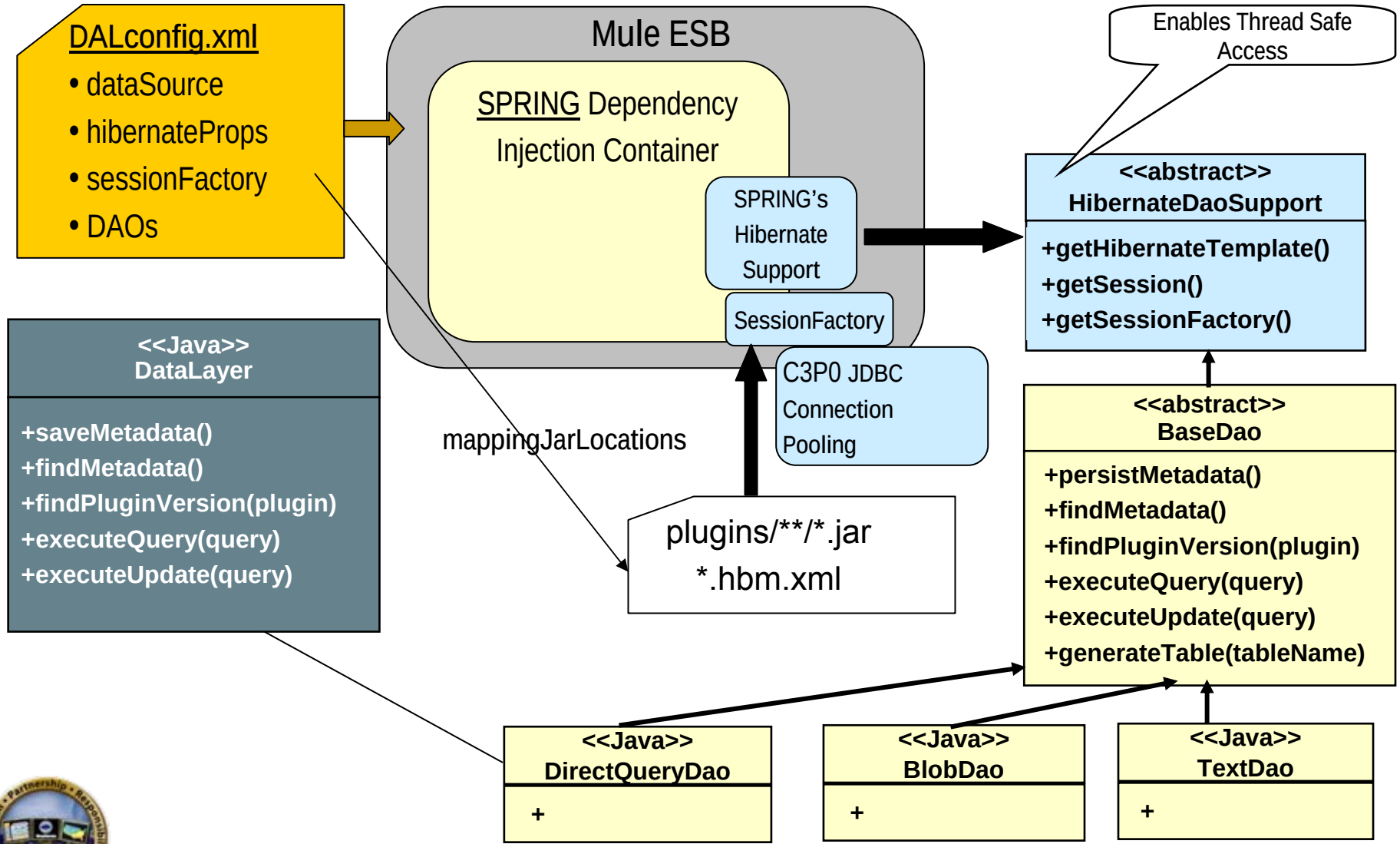
Data Object Model Extended By Plug In Follows Base Model

Plug-In Enables Adapting to New Data Types

ADE Data Access Pattern
Layered API Leveraging Spring's Hibernate Support

Raytheon

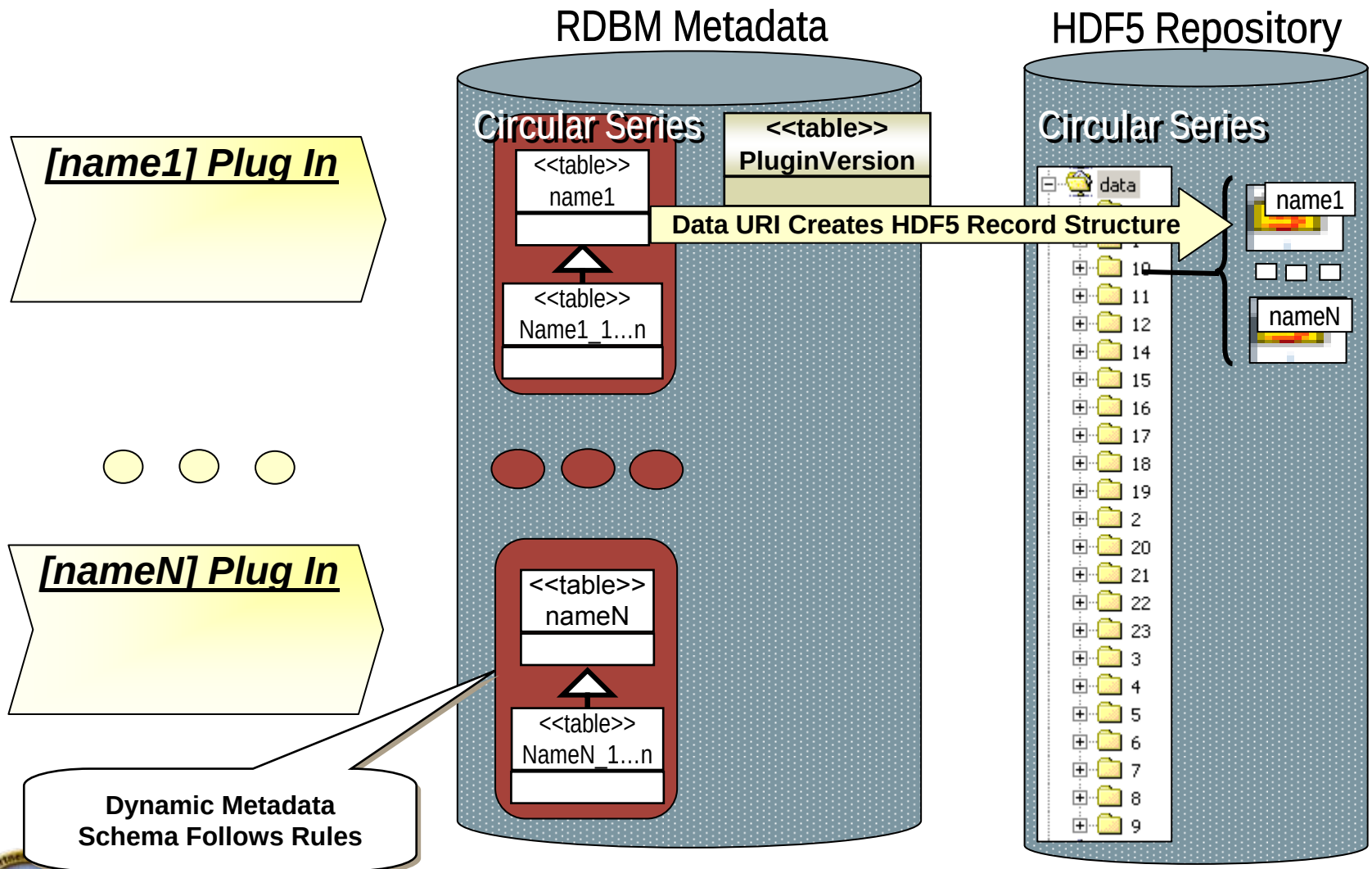
Data Model



SOA Plug-In Defines a Metadata Table Set -- Each Plug-In Also Defines an HDF5 Set

Raytheon

Data Model

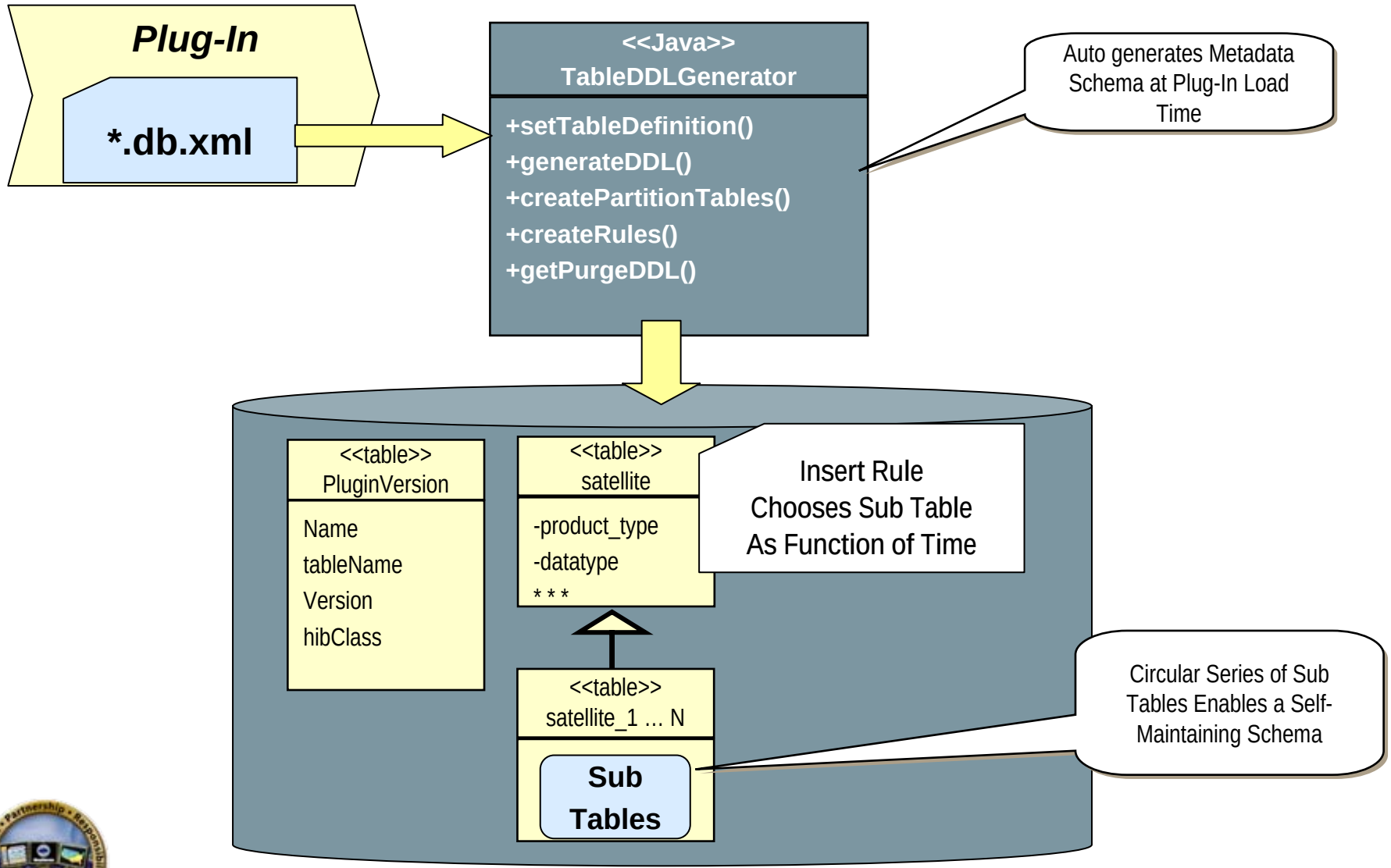


Flexible Data Model Is Plug-In Extendable to New Data Types

Plug-In Creates New Metadata in RDBMS
Uses PostgreSQL Table Inheritance and Rules

Raytheon

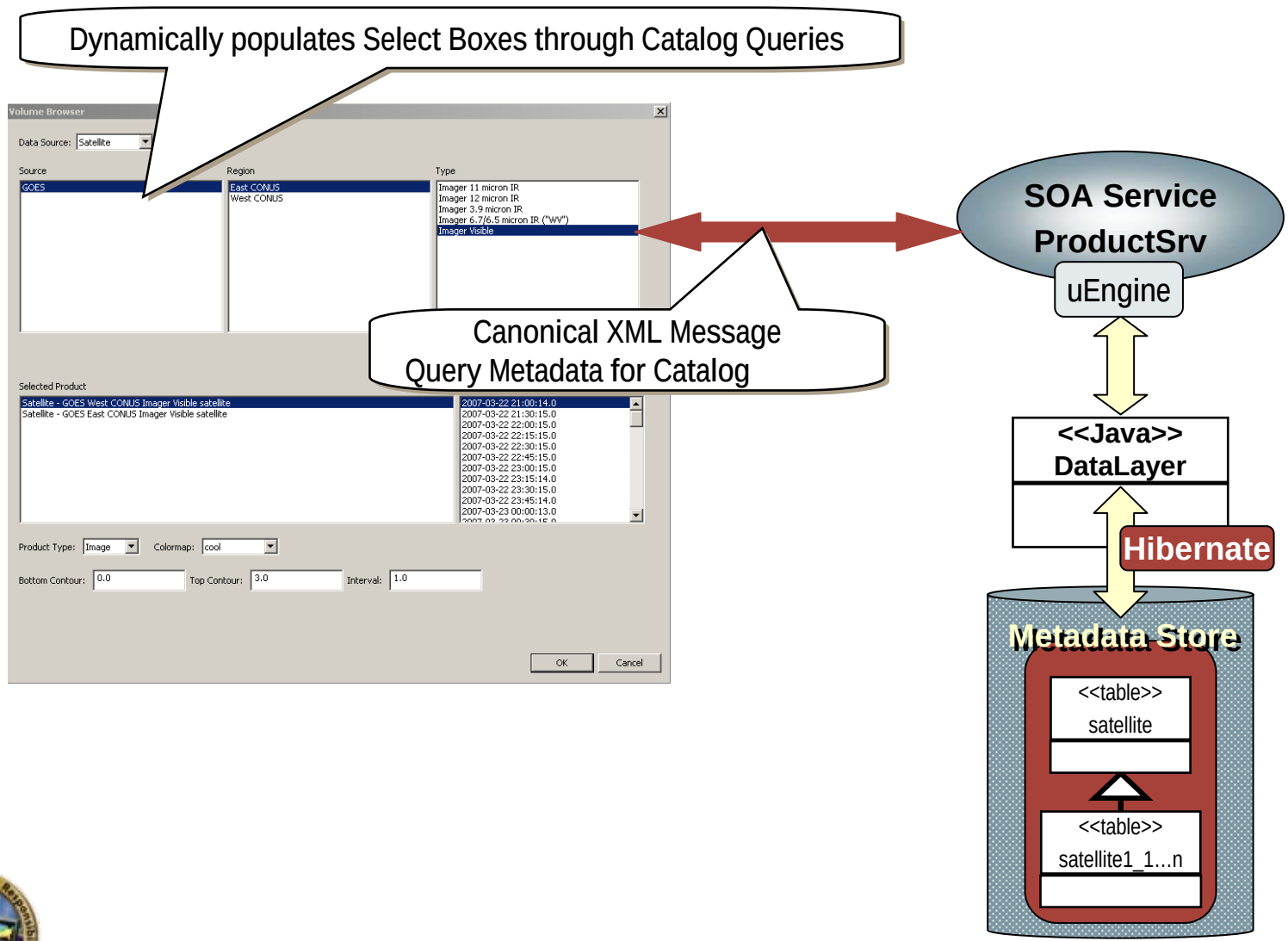
Data Model



Metadata Demo – Using CAVE's Volume Browser

Raytheon

Data Model



General DataURI Concept –
Key for System Adaptability to New Data Types

Raytheon

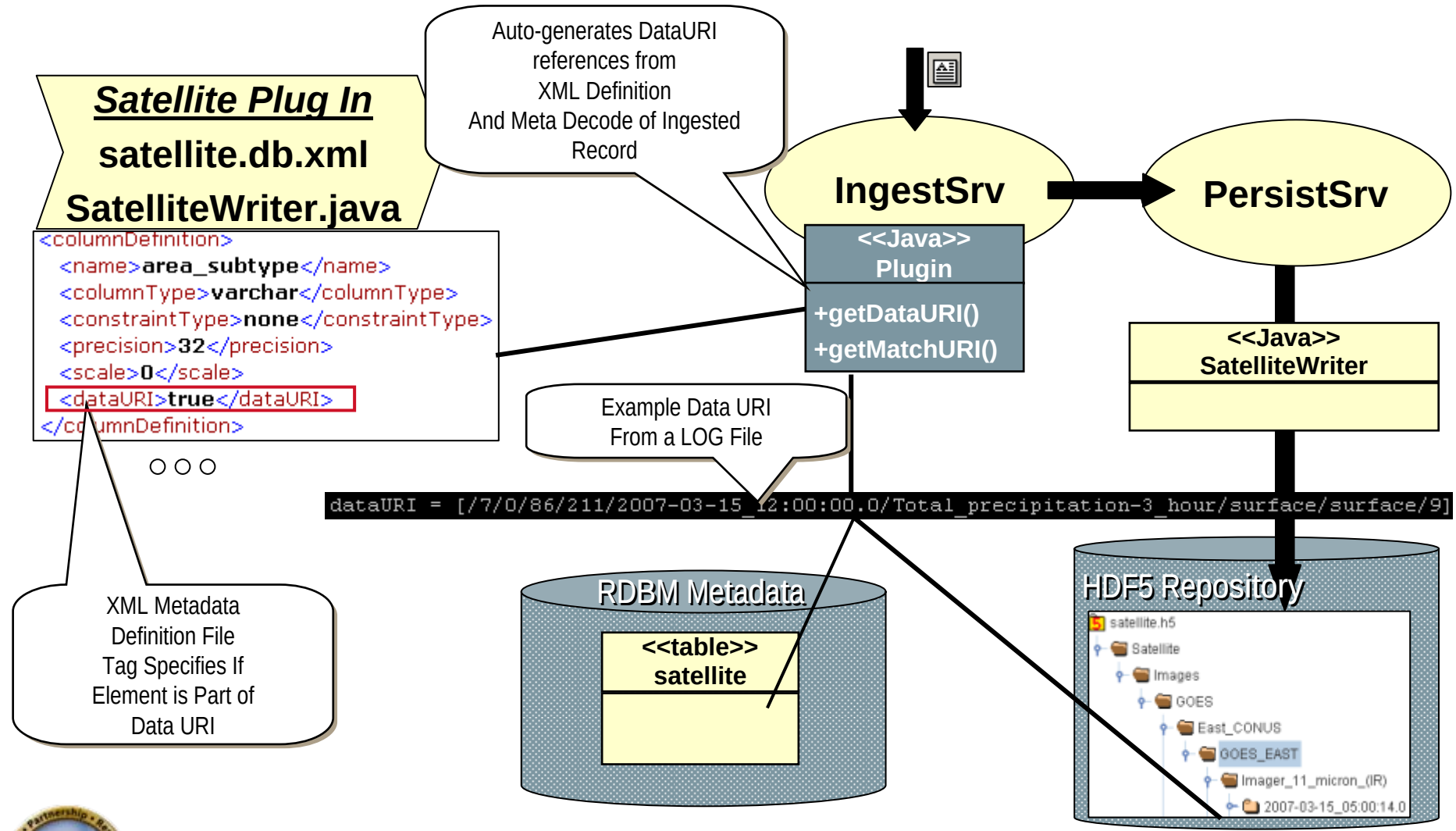
Data Model

- DataURI: a reference to data in the data store (i.e., D2D Data KEY)
- Enables Automatic Subscriptions For all Ingested Data
- Automatically ties data persistence to meta data
- Enables Plug-In Extendibility to new data types without changing any base code
- ADE implemented a design for automatic generation of DataURIs

Metadata Model Drives DataURI
Auto-Generated DataURI Couples HDF5 to Metadata

Raytheon

Data Model



Plug-In Enables Adapting To New Data Types

Data Persistence Using HDF5
HDF5 Files In Time-Ordered Bins Like Metadata

Raytheon

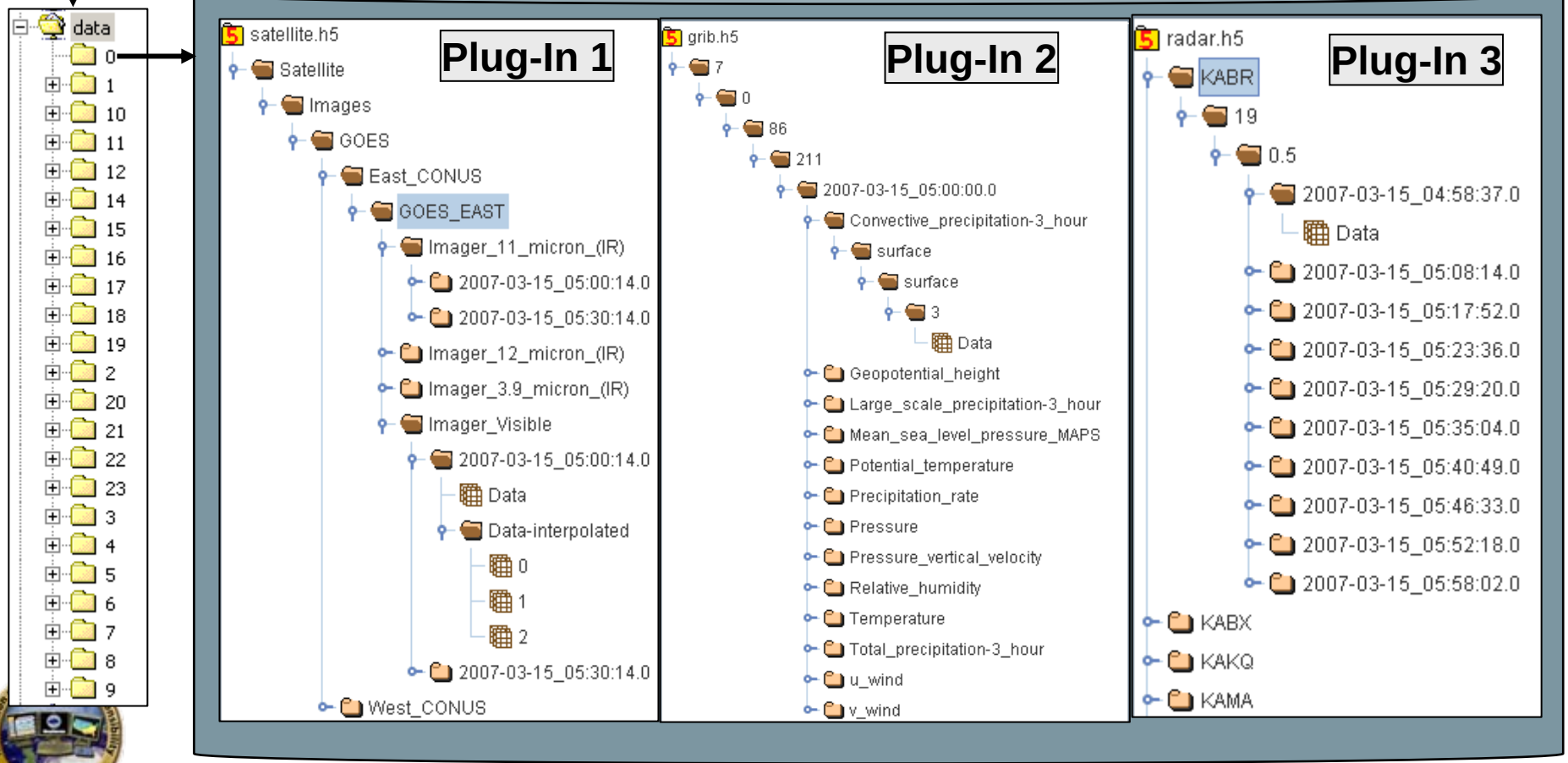
Data Model

Circular Time Bins

/awips/opt/data/hdf5

i.e., Auto-generated dataURI ties Meta Data to HDF5 Record

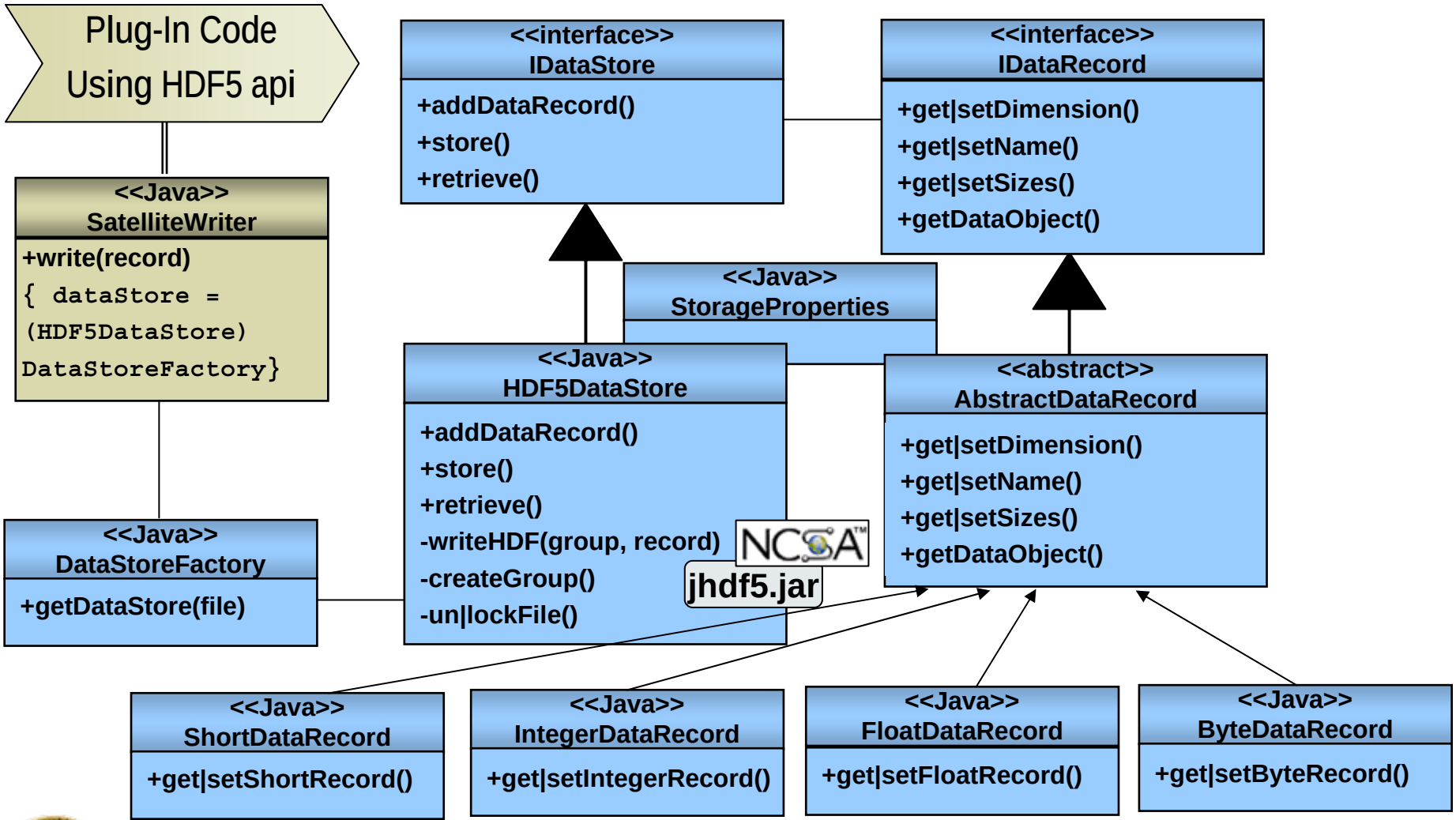
```
dataURI = [/7/0/86/211/2007-03-15_12:00:00.0/Total_precipitation-3_hour/surface/surface/9]
```



ADE Data Persistence Using HDF5
Application Code Interfaces Through API

Raytheon

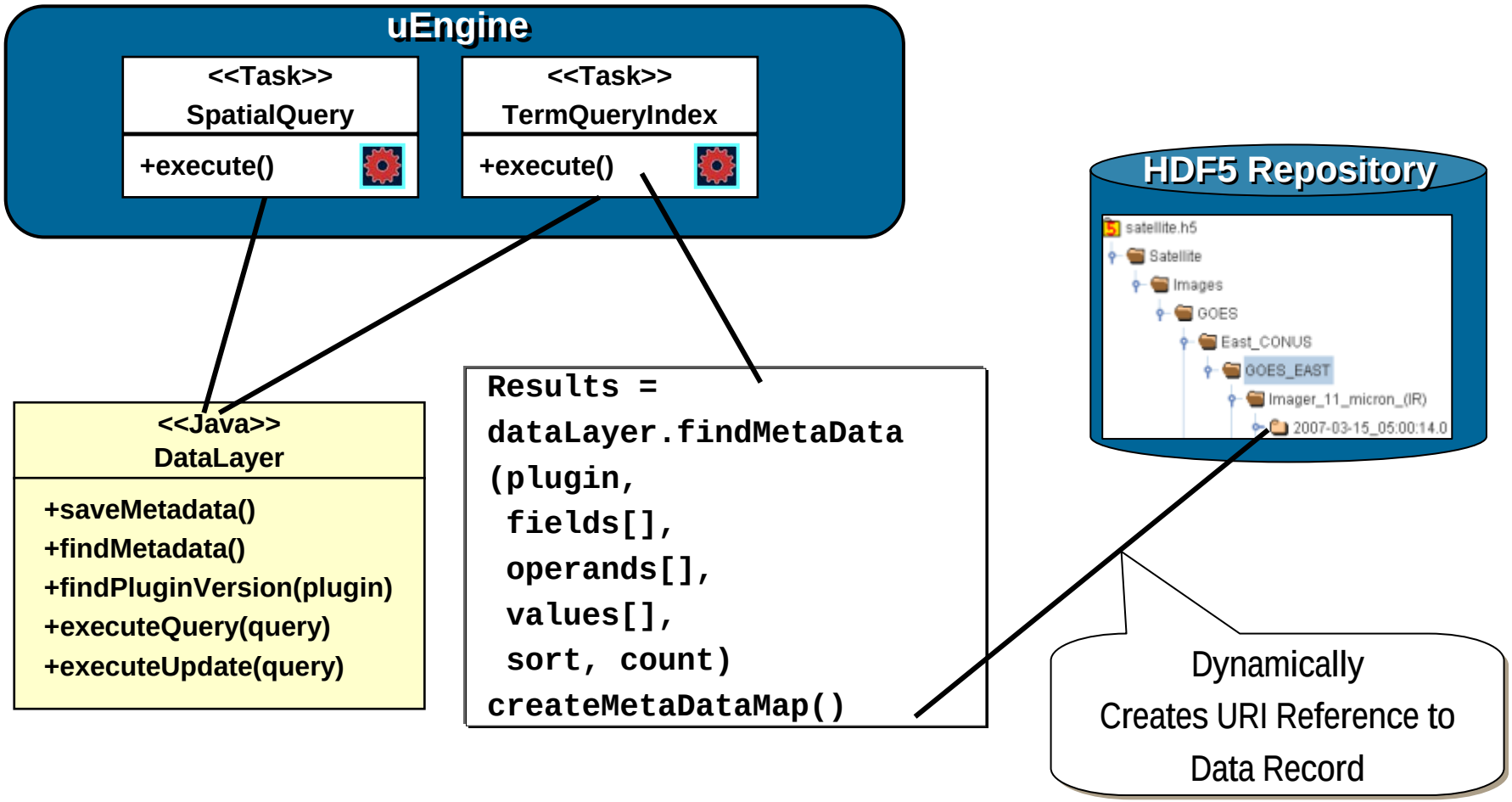
Data Model



uEngine Using the Data Access Layer
Single API Enables uEngine to Access All Data

Raytheon

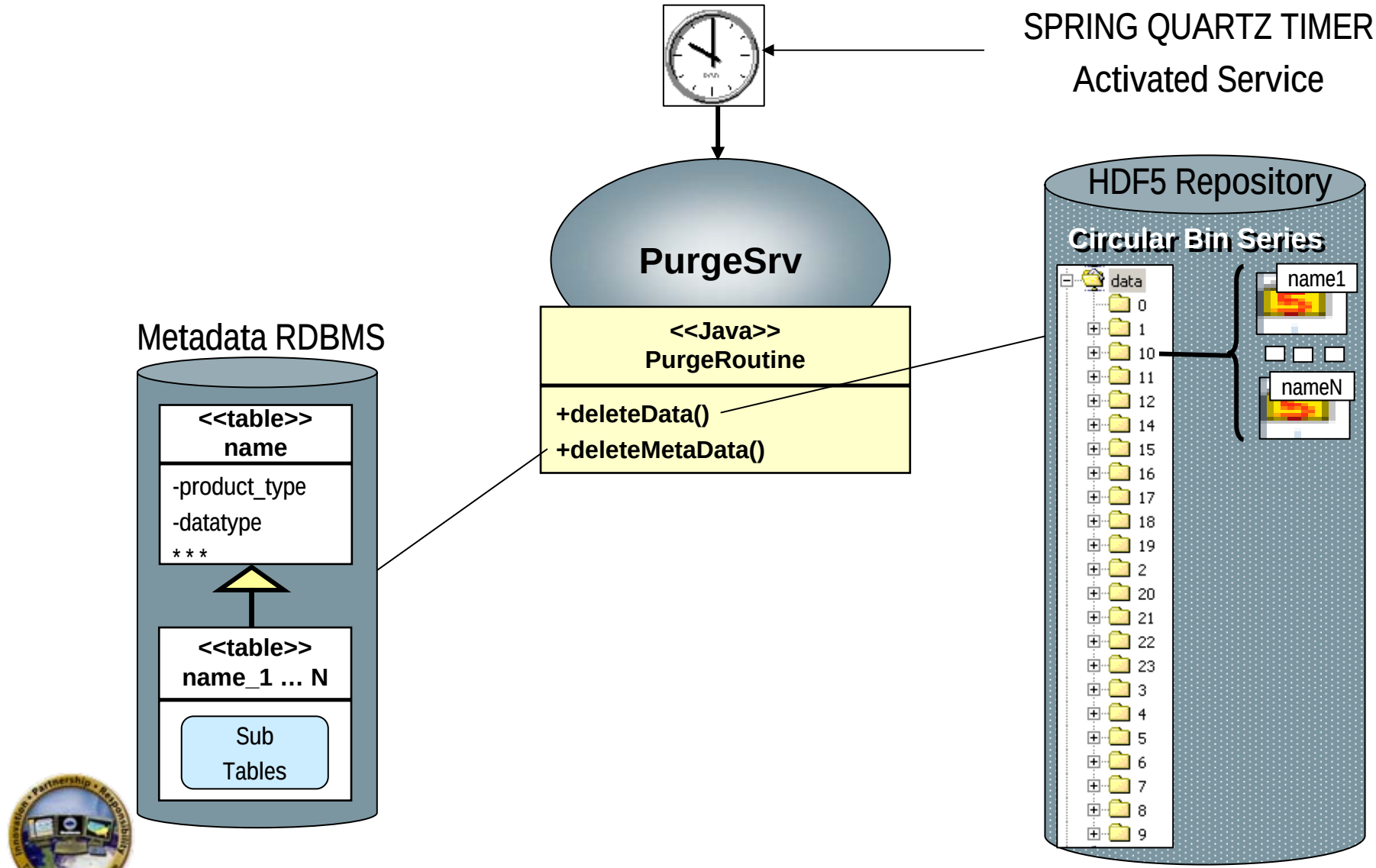
Data Model



Purging Data: Self-Maintaining
Drops Metadata Tables & HDF5 Bins Periodically

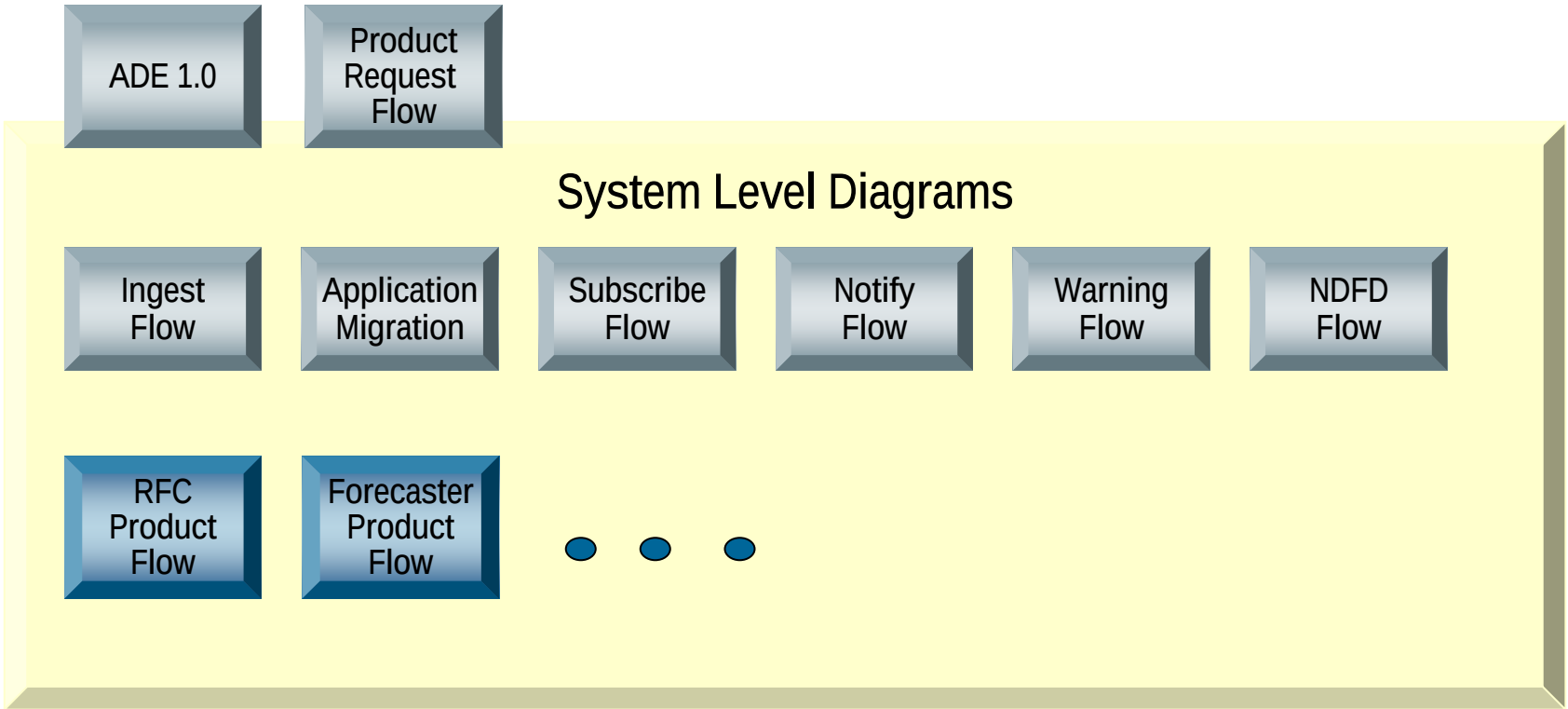
Raytheon

Data Model



System Flow Diagrams

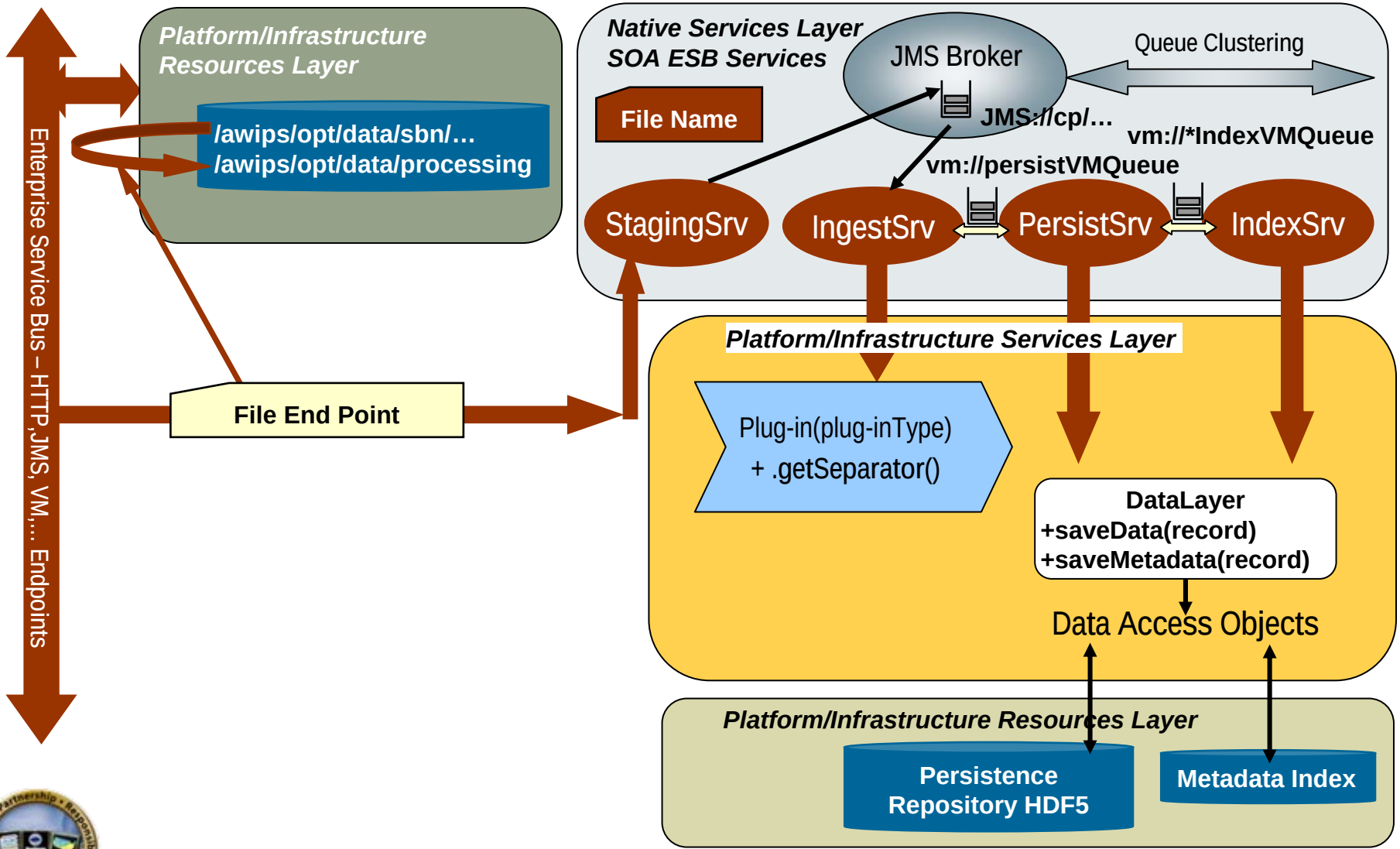
Service Flow



Ingest Flow
Ingest at a Clustered End Point

Raytheon

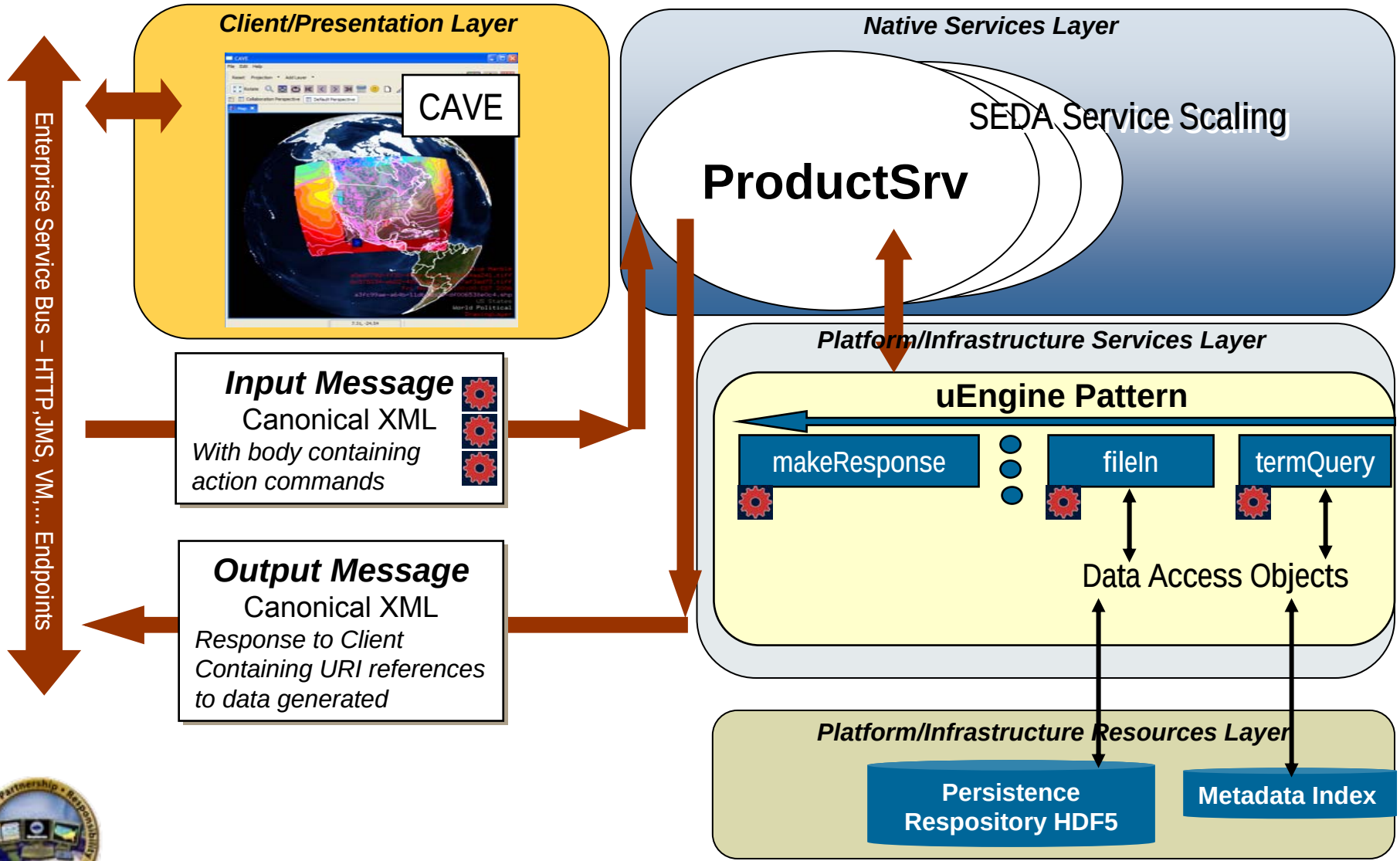
Service Flow



Product Request Flow
Cave Requests Data for Display as a GIS Layer

Raytheon

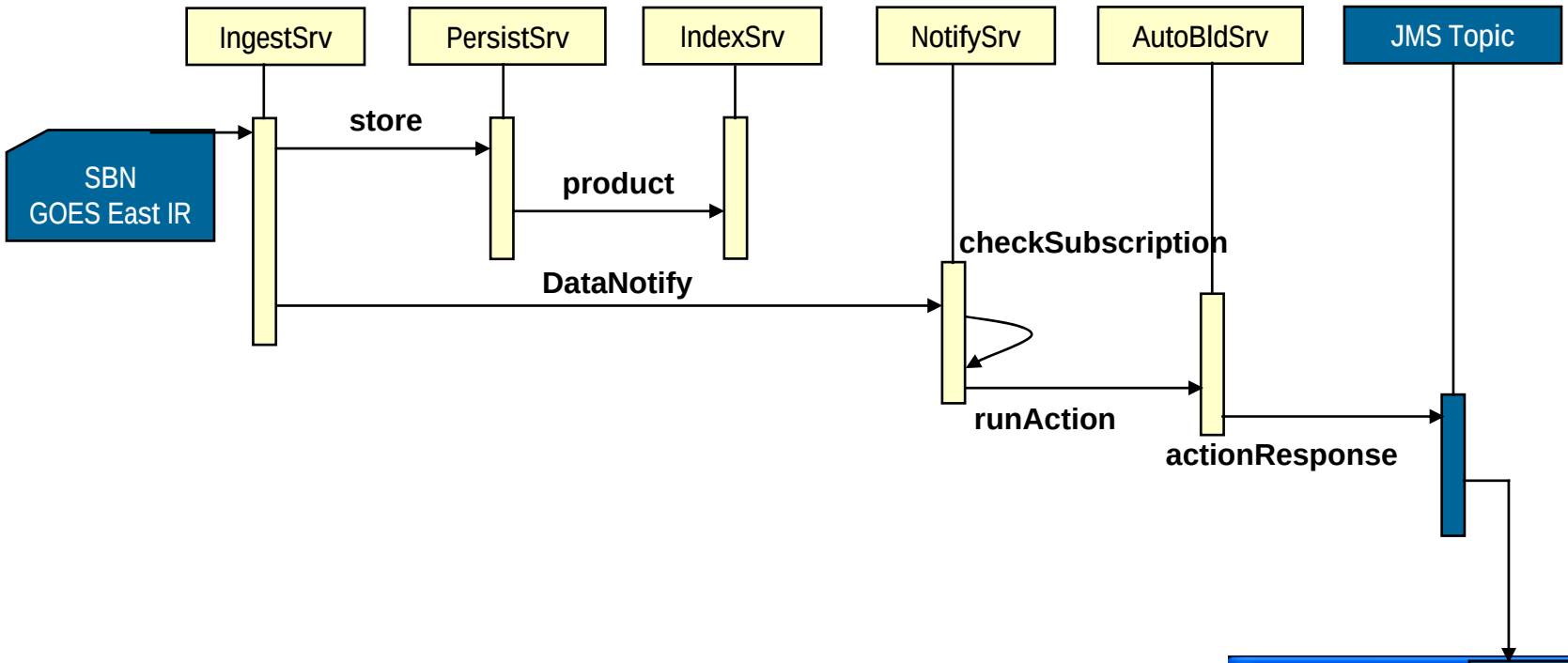
Service Flow

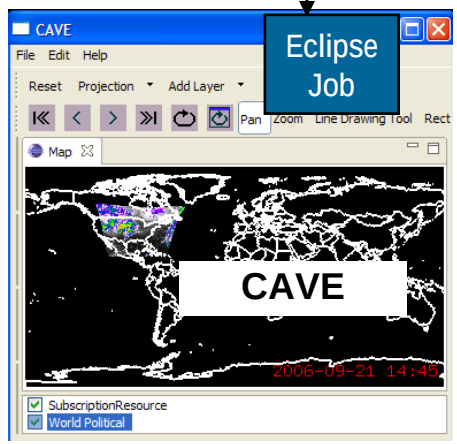


Notification Flow – Ingest Flow Triggering Notification

Raytheon

Service Flow



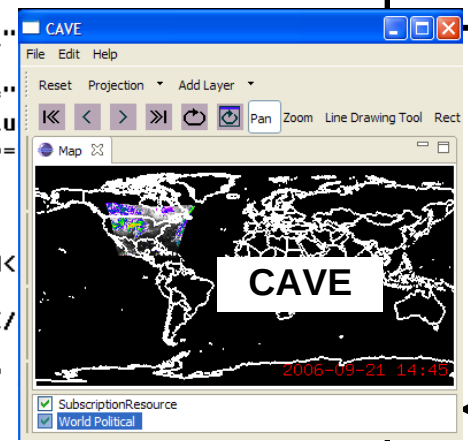


Subscription Flow – CAVE Requests a Subscription

Raytheon

Service Flow

```
<message>
   <header>
      <property name="id" value="subscriptiontest" />
      <property name="time" value="20060912162636" />
      <property name="function" value="execute,subscribe" />
   </header>
   <body>
      <action name="Sample Image Retrieval"
         <termQuery>
            <query name="datatype" value="
            <query name="locationkey" valu
            <query name="parameter" value=
         </termQuery>
         <fileIn />
         <decodeIndexedImage />
         <colorImage><colorMap>IREnhanced<
         <reproject/>
         <imageOut><format>tiff</format></
         <fileOut />
         <makeResponse returnMethod="uri"
      </action>
   </body>
</message>
```



BREAK


Innovation • Partnership • Responsibility
The Raytheon Team's Team

Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 2: Installation, Build, and Regression Test (Multiplatform) (rev. 1)

February 14, 2008

AWP.TRG.SWCTR/TO6.ADE/CAVE02.01

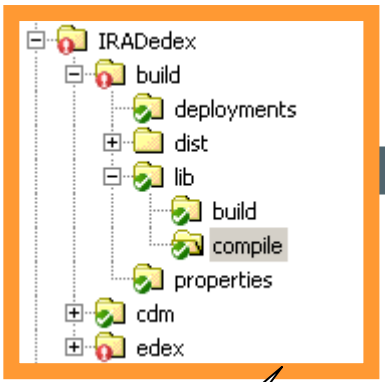
Objectives

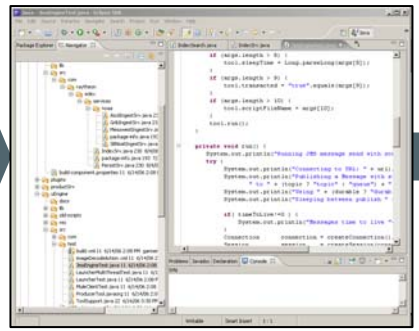
- Successfully install ADE
- Have ADE ready for running and development
- Successfully do a “Clean Build and Deploy”
- Successfully verify system installation by running a Standard Regression Test through a Regression Test GUI (e.g., Tomcat)
- Learn how to use Debug to step through code running in services

AWIPS Development Environment (ADE) – An End-to-End Technical Reference Architecture

Raytheon

CM Source Repository | Eclipse IDE for Java | Build Products | Deployment





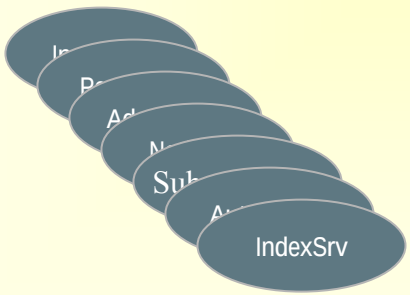
ADE Build Process

Workstation

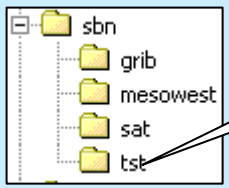
Documentation
- JavaDoc
- Design Docs
- How-to ...

Server/ Workstation

Service Container



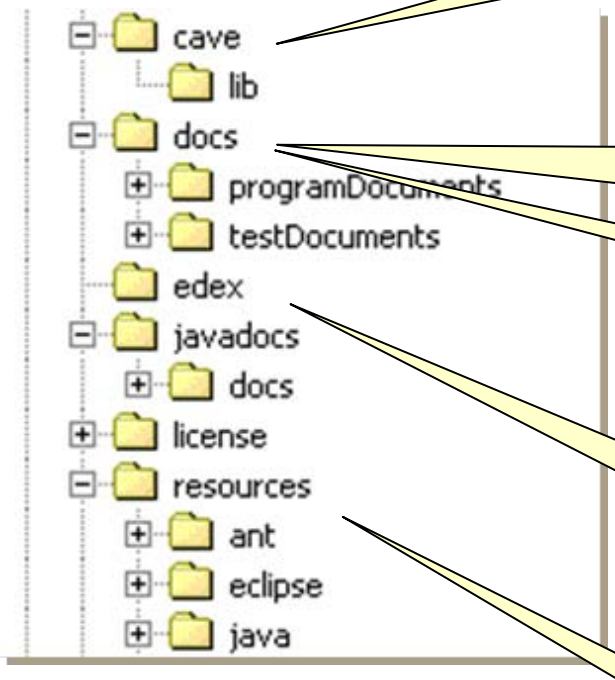
File System Endpoints



Regression Tests Data Sets

Persistent Store | Metadata Index | Spatial Index

ADE Delivery: One DVD Posted to Site



CAVE Install
- Source as Eclipse project
- Maps and sample data

Documentation
- Software Test Plan
- Requirements Traceability Matrix
- uEngine scripts

ADE Install Procedure
- Linux
- Windows

Server Side Install (EDEX)
- Execution environment
- Self-extracting deployment
- Regression test data
- Source as Eclipse project

Dependencies
- ANT
- JAVA + JAI
- Eclipse IDE

ADE Install Procedure

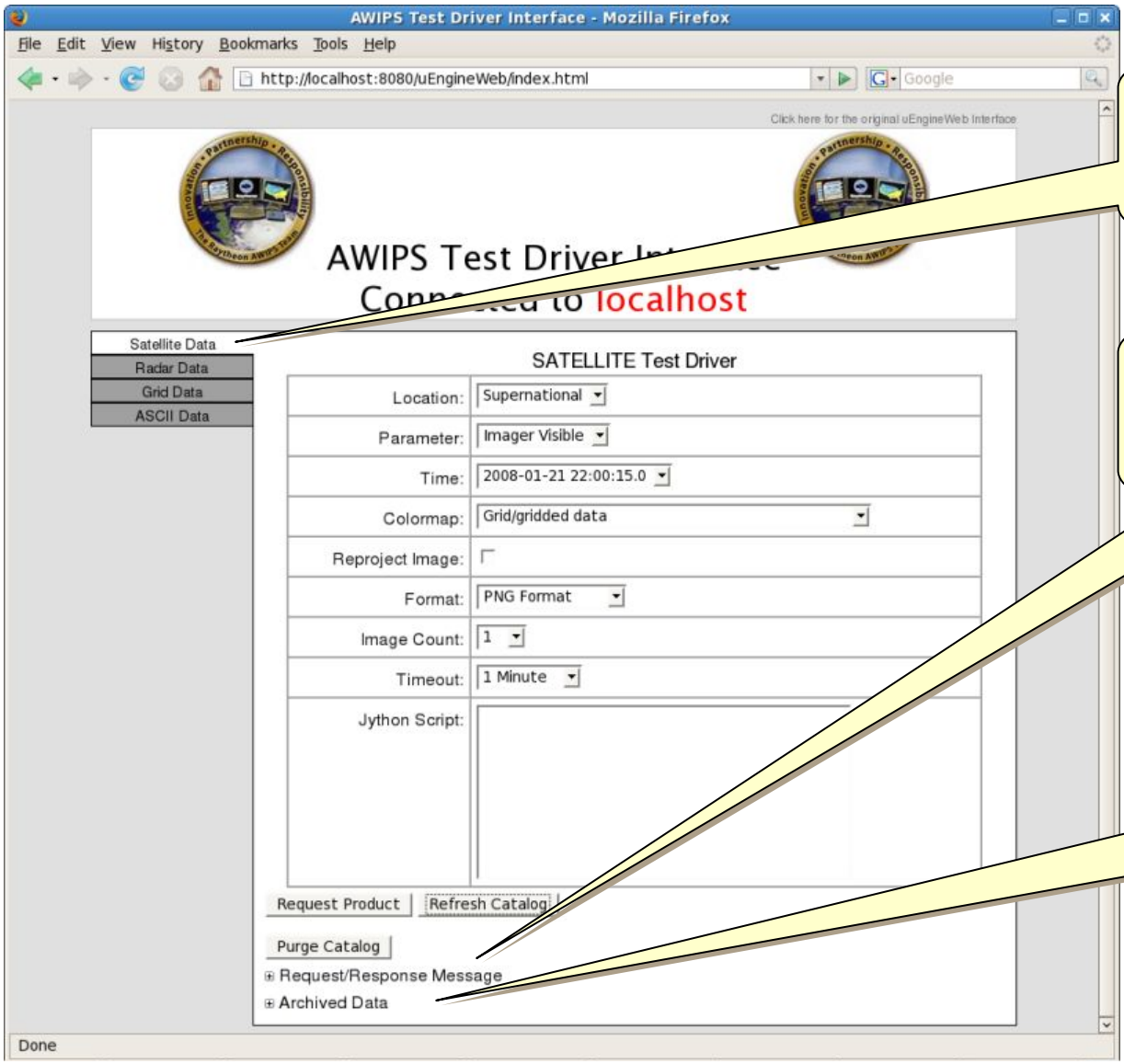
ADE installation instructions are documented in Module 9: Installation/Deployment.

Start-Up Server Side

- RHEL Linux
  - Start ActiveMQ: awips/activemq/bin/activemq-standalone.sh
  - Start Mule: awips/opt/esb/bin/start.sh standalone
  - Start Tomcat: awips/tomcat/bin/startup.sh

- Windows
  - Start the VMWare image
  - Start ActiveMQ: awips/activemq/bin/activemq-standalone.sh
  - Start Mule: awips/opt/esb/bin/start.sh standalone
  - Start Tomcat: awips/tomcat/bin/startup.sh

ADE Regression Tests: Start Tomcat



Select a Data Type
- Performs a catalog query

View JavaScript Messages
- Editing the request

Ingest Regression Test Data
- Performs the copy to ESB endpoint

ESB / Container Log File for Ingest

```
INFO  2006-12-21 10:34:22,515 [SBN-Radar-Ingest.3] noaa.SBNradarIngestSrv: RadarIngestSrv receiv
ed a message with: ..\..\data\sbn\radar\KMOB_SDUS54_N0REUX_200610271705len: 31848^M
INFO  2006-12-21 10:34:22,515 [SBN-Radar-Ingest.3] noaa.SBNrada                Ingesting messa
e: ..\..\data\sbn\radar\KMOB_SDUS54_N0REUX_200610271705 in 0.0s^M
INFO  2006-12-21 10:34:47,250 [ProductSrv.4] services.ProductSrv: Received Product Req
t, len: 352^M
<message>
<header>
<property name="id" value="Catalog" />
<property name="time" value="20061221103447" />
<property name="function" value="execute" /></header>
<body>
<action>
<catalog>
<queryType>distinctValue</queryType><queryField>product_code</queryField><constraint na
al_descriptor" value="radar"/></catalog>
</action>
</body>
</message>
^M
```

Radar Ingest Service
- Note 3rd instance

Catalog Query
- To ProductSrv
- Canonical XML msg

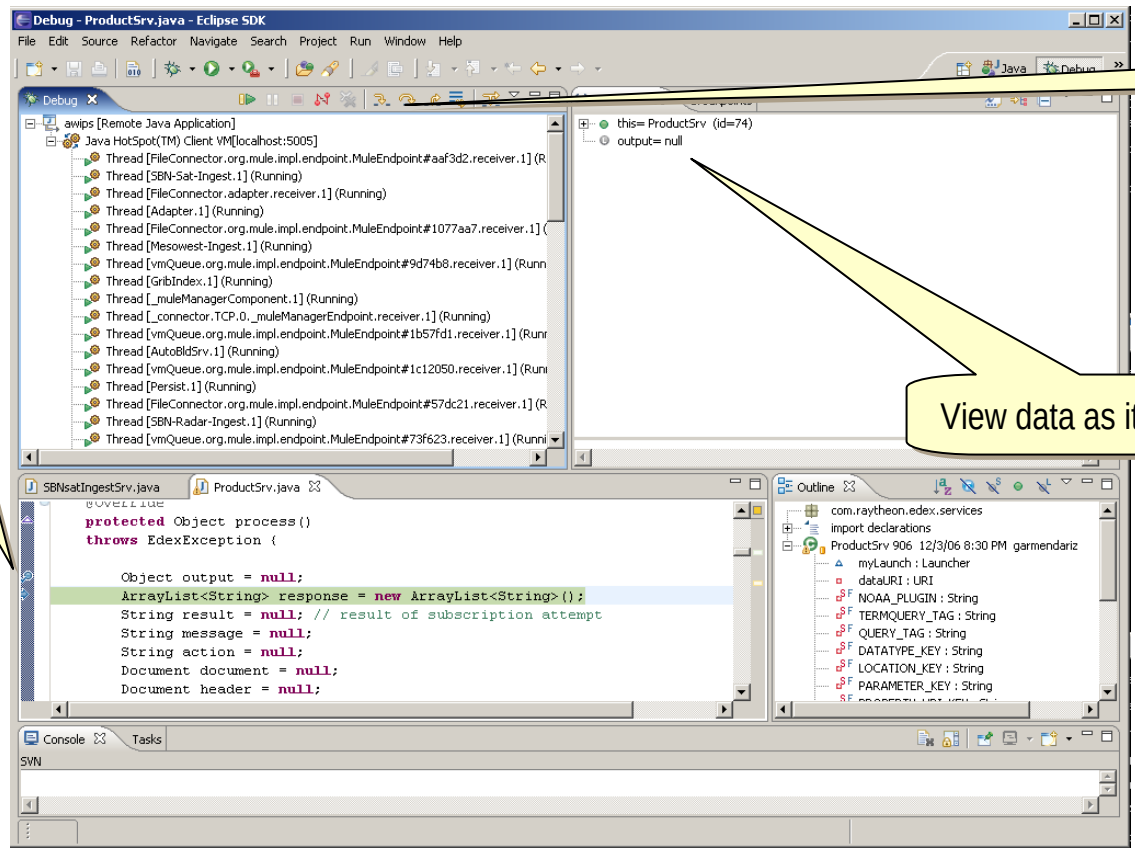
Remote Debugging of ESB SOA Services Example Stepping Through "ProductSrv"

Raytheon

- Enable debug in awips/mule/conf/wrapper.conf

```
# uncomment this line to debug remotely, the application will wait for the external debug
wrapper.java.additional.3=-Xdebug
wrapper.java.additional.4=-Xnoagent
wrapper.java.additional.5=-Djava.compiler=NONE
wrapper.java.additional.6=-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005
```

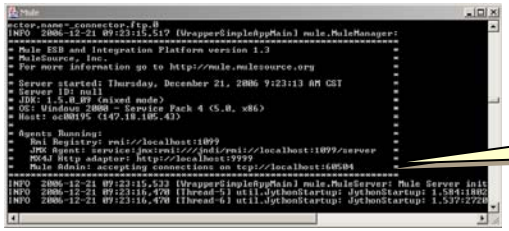
Right click to set breakpoint at beginning of +process() method



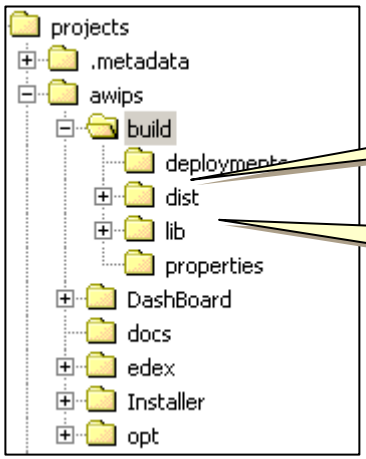
Step controls

View data as it changes

Server Side: Developer Build and Deploy

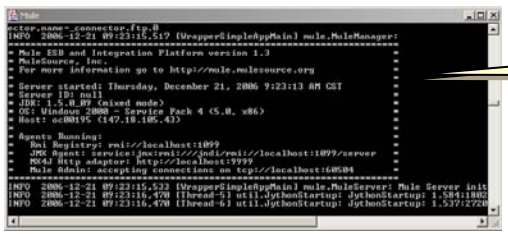


Shutdown server processes: Ctrl-C in command window. Shutdown hook ensures clean shutdown.



Change directory to build directory in your project area using a command terminal.

ant clean	removes “dist” directory
ant build	performs a smart build
ant jibx	creates the JiBX bindings
ant deploy	deploys build artifacts to run environment
ant uEngineWeb	regression test web site



Start server processes and test.

Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 3: MicroEngine Scripting (rev. 2)

February 14, 2008

AWP.TRG.SWCTR/TO6.ADE/CAVE03.02

Prerequisites/Objectives

- Prerequisites
  - Access to an installation of ADE 1.0
  - Familiarity with utilizing Eclipse for Java development
  - Familiarity with Object Oriented Programming
  - Programming experience in the Java programming language
    - ADE 1.0 utilizes Java 1.6
- Objectives
  - Understand how to create tasks and scripts for the MicroEngine (uEngine)

Estimated Time: 1 hour

uEngine Overview

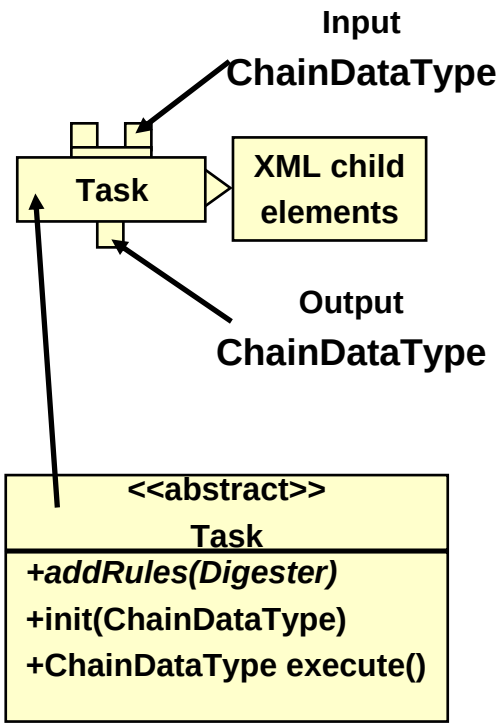
- Architecture
- Tasks
- JavaScript Scripting
- Client Applications

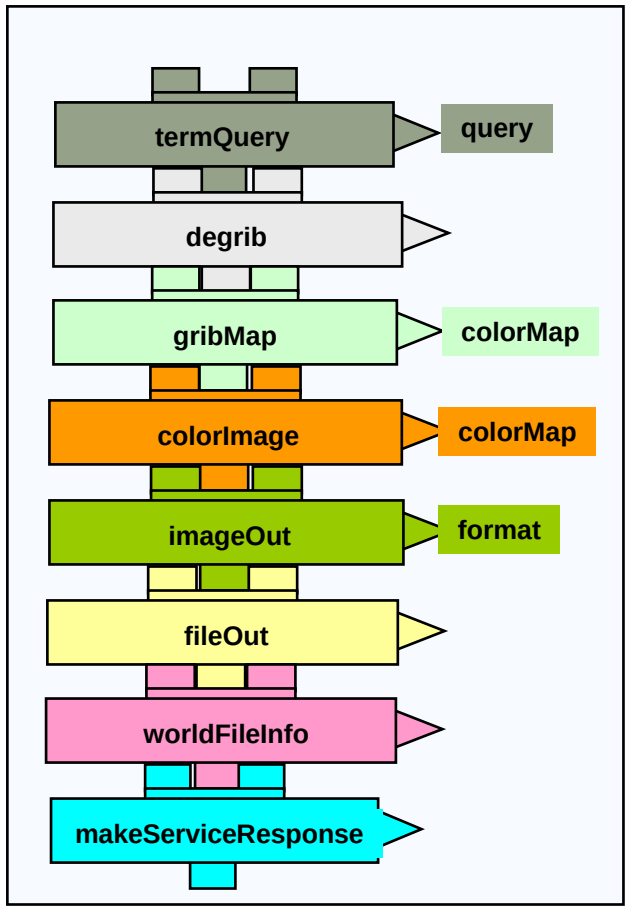
uEngine Task Execution Pattern
Breaks Up Execution Into Small, Reusable Tasks

Raytheon

- uEngine Vision
  - Create an execution framework for generating custom products on demand
  - Customer systems can request products by script requests over a network
  - Script performs small general units of work that get chained together to produce a customer product

uEngine Task Execution Pattern
Original Concept – XML-Based Declarative Scripts





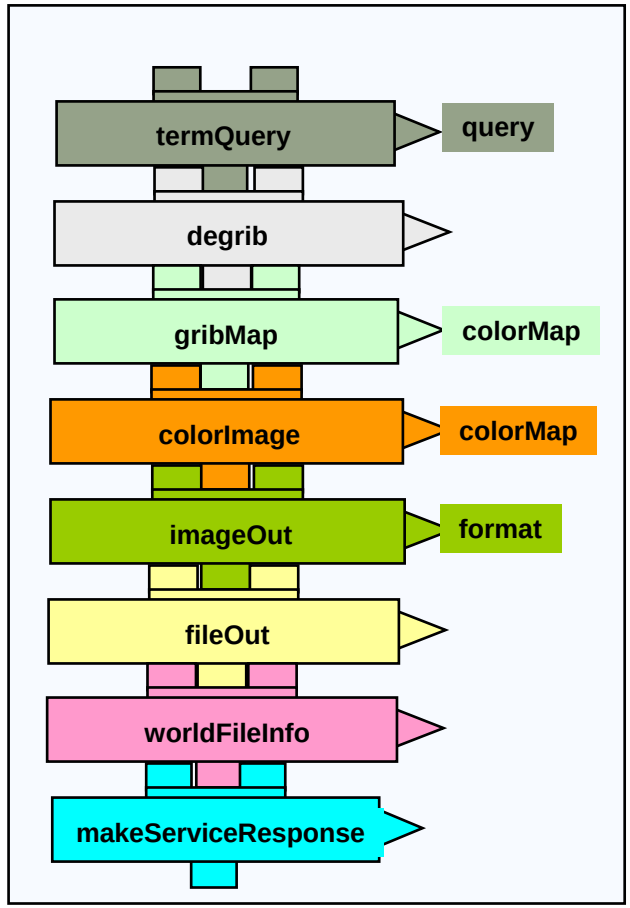

```
<action name="ServiceGribImageDecode">
   <termQuery>
      <query name="grid_type"
             value="22" />
   </termQuery>

   <degrib/>

   <gribMap colorMap="GribRGB" />

   <colorImage colorMap="GribRGB" />

   <imageOut format="png" />

   <fileOut/>

   <worldFileInfo/>

   <makeServiceResponse />
</action>
```

Adding a new uEngine Task

1. Extend Task
2. Implement addRules()
3. Implement abstract methods

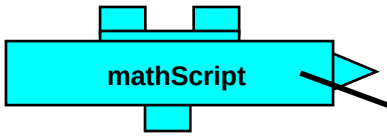
uEngine Task Execution Pattern
Original Concept – XML-Based Declarative Scripts

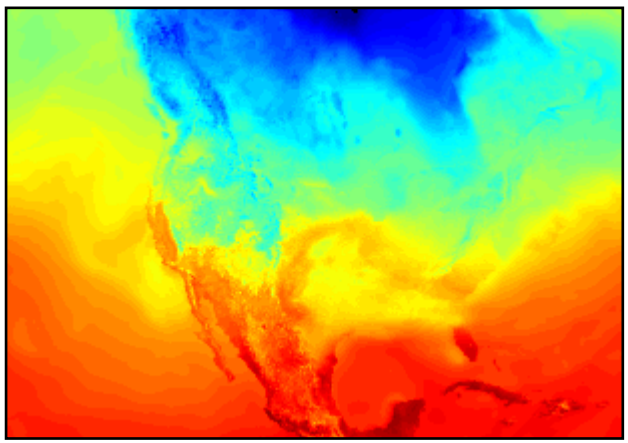
- µEngine scripts seen as a series of blocks needed to build a visualization product
  - Scripts were seen as following a simple I-P-O (input, process, output) flow of control
  - All tasks processed the **same set of data**
- Most of the "grunt" work performed within the tasks
  - Looping and conditional processing was built into the individual tasks
  - Passing information between tasks became very complicated
- Result: Scripts were ***fairly easy*** to write; tasks were ***extremely difficult*** to write



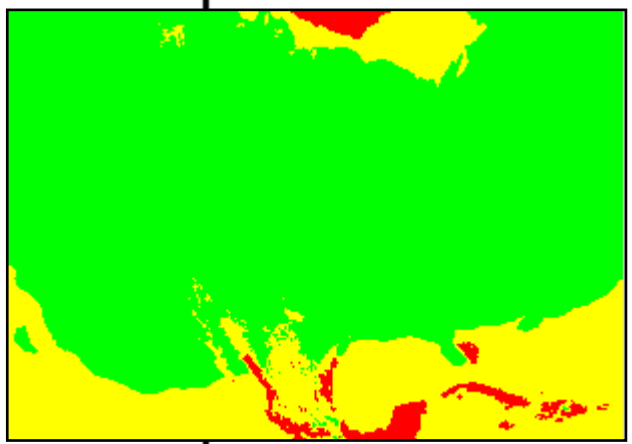
uEngine Task Execution Pattern
Data Transformation Into Decision Aids: Scripting

Raytheon



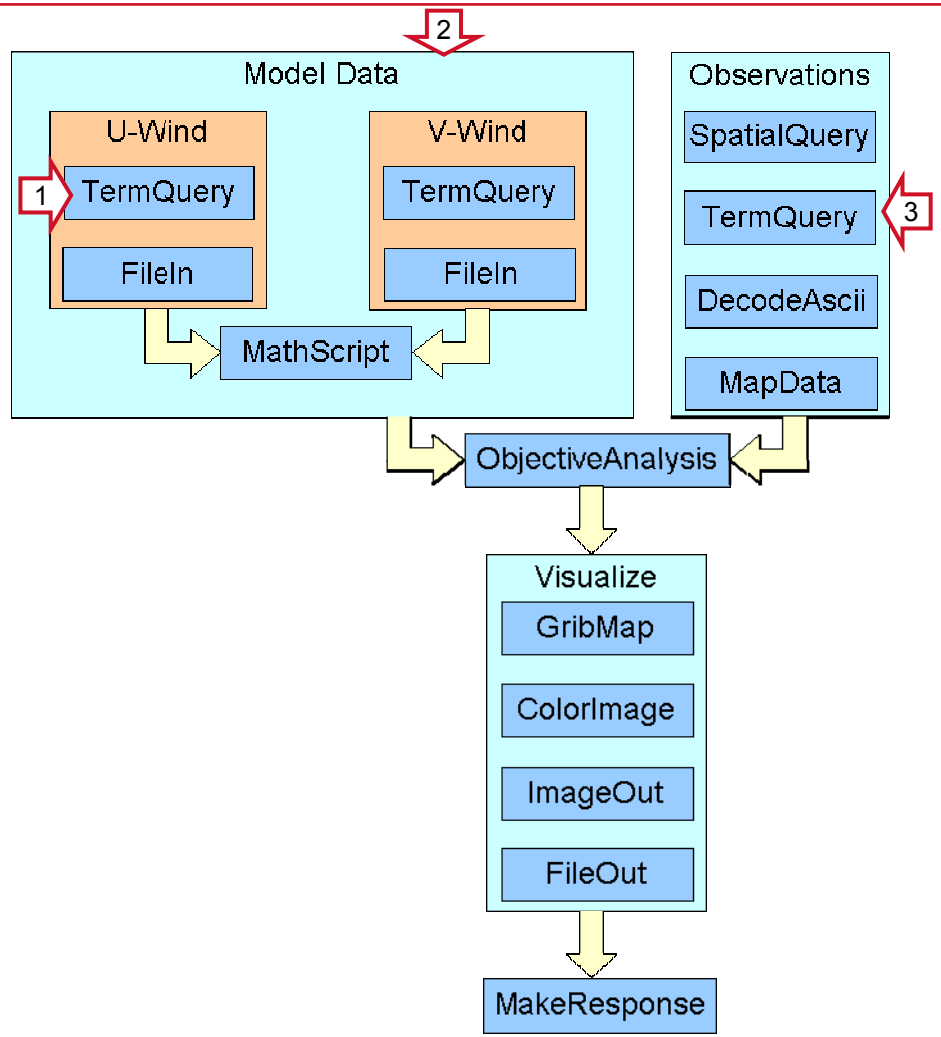


```
<action name="GribImageDecode">
   <termQuery>
      <query name="parameter_type" value="TMP" />
      <query name="first_fixed_value" value="2" />
      <query name="forecast_time" value="3" />
   </termQuery>

   <degrib />

   <mathScript>
      <input name="STDIN" value="default" />
      <param name="NULL" value="9999" />
      <![CDATA[
import com.raytheon.edex.uengine.util.jython as utilities
utils = utilities.JMath()
def trans(z):
   if z == NULL: return red
   elif z < p1:  return red
   elif z < p2:  return yellow
   elif z < p3:  return green
   elif z < p4:  return yellow
   else:         return red
red = 2.0
yellow = 1.0
green = 0.0
MIN = utils.amin(STDIN)
MAX = utils.amax(STDIN)
rng = MAX - MIN
p = (1.0 * rng) / 10
p1 = MIN + p
p2 = p1 + p
p3 = p2 + 6 * p
p4 = p3 + p
STDOUT = [trans(i) for i in STDIN]
]]>
   </mathScript>

   <gribMap colorMap="StopLight" />

   <colorImage colorMap="StopLight" />

   <imageOut format="png" />

   <fileOut />
   <worldFileInfo />

   <makeResponse returnMethod="uri"/>
</action>
```



uEngine Task Execution Pattern
Original Concept: XML-Based Declarative Scripts

- More complicated products require more complicated data and logic flows
  - The diagram shows the data flow used to create a wind speed product
  - Separate data flows for model and observation data combine to form a single visualization product
- This requires the ability to
  - support multiple data queries (1)
  - selectively process the data (2)
  - process multiple return values from a query (3)
- These considerations drove the need for a *better* scripting engine.



uEngine Task Execution Pattern Scripting Engine Alternatives: Jython

Raytheon

- Jython
  - Pure Java implementation of Python
  - Current implementation Version 2.2b2 – release date: May 10, 2007
    - Note that Python is currently at release 2.5.1, released April 18, 2007
    - Python 2.2.3 was released on May 30, 2003, so Jython is essentially 4 years behind Python
    - Development of Jython support products also lags behind Python
  - Numerous books on Python/Jython programming available
  - Jython integrates well with Java

uEngine Task Execution Pattern Scripting Engine Alternatives: JavaScript

Raytheon

- JavaScript/Rhino
  - JavaScript: A Java-like scripting language originally developed for Web browsers (Netscape 2)
  - Rhino: an Open Source implementation of JavaScript written in Java
    - Does not require Web browser for execution
  - Numerous books on JavaScript programming available
  - Rhino integrates reasonably well with Java
- Java Scripting API
  - Java SE 6 includes the JSR 223=based Java Scripting API
    - Defines a consistent interface for embedding script engines in Java apps
    - The only “out of the box” scripting support in Java SE 6 is for a Rhino-based JavaScript engine
- Rhino-based JavaScript: The scripting language of choice

Architecture

- Uses Mozilla's freely available Rhino
  - Same JavaScript engine in JDK 1.6, except that Mozilla version offers more features
  - Used in Eclipse
- UEngineScript class – simply call setScriptText(String javaScriptCode) and run()
- Subscription scripts compiled once for faster execution
- Can include .js libraries in a script

Architecture: How It Works

- Rhino returns the last executed global statement of the script
  - That is, the last statement not in a function
  - Sample Query for METARS Script (Slide 83) returns the execution of metarQuery();
  - UEngineScript will convert this to an Object[]
- Because tasks should only operate on one data object, “for” loops are used in the JavaScript
- queryResults is a java.util.ArrayList object; Java objects can be used in the JavaScript script
- Not forced into chain data; instead pass along the results of each task execution

Architecture: Potential Issues

- Loosely typed
  - Like most “scripting” languages, JavaScript is loosely typed
  - Cannot distinguish between the various numeric types
  - **Solution:** Rather than primitives, pass Java objects between Java and JavaScript
- No primitive arrays
  - JavaScript arrays are objects
  - Processing a Java array in JavaScript corrupts the object
  - JavaScript able to process java.util.List and java.util.Map based structures
  - **Solution:** Rather than arrays, pass java.util.List and java.util.Map based structures between Java and JavaScript

Note: *Starting with Java SE 5, Java automatically converts between primitives and objects as needed.*

Tasks

- Tasks extend ScriptTask
  - Tasks are “Plain Old Java Objects” (POJOs) with an execute() method
  - Tasks only operate on one object

```
public class DataToXml extends ScriptTask
{

    private AbstractDataRecord record;

    public DataToXml(Object anAbstractDataRecord)
    {
        record = (AbstractDataRecord) anAbstractDataRecord;
    }


    /* (non-Javadoc)
     * @see com.raytheon.edex.uengine.js.tasks.ScriptTask#execute()
     */
    @Override
    public Object execute()
    {
        return Util.marshalToXMLString(record, record.getClass());
    }

    public AbstractDataRecord getRecord()
    {
        return record;
    }


    public void setRecord(AbstractDataRecord aRecord)
    {
        record = aRecord;
    }

}
```

Programming Example

- Create an Engine Task that writes a message to the system log file
  - Logger will support 3 levels; “warning,” “info,” and “debug”
  - Logger will have a single method for logging a message

```
/* partial JavaScript example */
function _execute() {
  var logger = new SystemLog();
    .
    .
    .
  logger.log("info","This is a test!");
    .
    .
    .
}
```

```
INFO  2007-06-14 21:15:29,170 [Awips.Edex.Service.ProductSrv.2]
com.raytheon.edex.services.ProductSrv: executing action script: JS Request
INFO  2007-06-14 21:15:29,342 [Awips.Edex.Service.ProductSrv.2]
com.raytheon.edex.uengine.MicroEngineTaskManager: creating MicroEngineTaskManager instance
INFO  2007-06-14 21:15:30,014 [Awips.Edex.Service.ProductSrv.2]
com.raytheon.edex.uengine.tasks.process.SystemLog: Performing initial query...
INFO  2007-06-14 21:15:30,451 [Awips.Edex.Service.ProductSrv.2]
com.raytheon.edex.uengine.tasks.process.SystemLog: Preparing response...
INFO  2007-06-14 21:15:30,467 [Awips.Edex.Service.ProductSrv.2]
com.raytheon.edex.services.ProductSrv: Request completed in: 1.313s
```

Use Eclipse to demo the class creation.

Programming Example: Java Code

```
package com.raytheon.edex.uengine.tasks.process;

import com.raytheon.edex.uengine.tasks.ScriptTask;

public class SystemLog extends ScriptTask {

    private String level = "info";
    private String message = "";
    @Override
    public Object execute() {
        if (level.equalsIgnoreCase("error")) {
            logger.error(message);
        } else if (level.equalsIgnoreCase("info")) {
            logger.info(message);
        } else {
            logger.debug(message);
        }
        return null;
    }
    public void log(String level,String message) {
        this.level = level;
        this.message = message;
        execute();
    }
}
```

Use Eclipse to demo the class creation – comments omitted.

JavaScript Scripting

Benefits

- Familiar syntax
- Object Oriented scripts
  - Reusable objects and methods
  - Easier to update
  - Abstract/hide code
- Decision aids
  - If statements, for loops, etc.
- More customizable scripts

JavaScript Scripting: Sample GriB to Image Script

```
gribRequest();

function gribRequest()
{
        var plugin = "grib";
        var query = new TermQuery(plugin);
        query.setCount(3);
        query.addParameter("paramid", "Temperature");
        query.addParameter("levelinfo", "50000.0_Pa");
        var queryResults = query.execute();
        var responses = new Array();

        for(i=0; i < queryResults.size(); i++)
        {
                var currentQuery = queryResults.get(i);
                var fileIn = new FileIn(plugin, currentQuery);
                var gribMap = new GribMap(plugin, "GribRGB", fileIn.execute(), currentQuery);
                var imageData = gribMap.execute();
                var colorMap = new ColorMapImage("GribRGB", imageData, gribMap.getGridGeometry());
                var format = "png";
                var imageOut = new ImageOut(colorMap.execute(), format, gribMap.getGridGeometry());
                var fileOut = new FileOut(imageOut.execute(), format);
                var makeResponse = new MakeResponseUri(fileOut.execute(), null, currentQuery.getDataURI(), format);
                responses[i] = makeResponse.execute();
        }

        return responses;
}
```

JavaScript Scripting: Sample Query For METARS Script

```
function MetarRequest(){}

function metarQuery(count)
{
                var query = new TermQuery("metar");
                query.addParameter("reportType", "METAR");
                query.setCount(count);
                var queryResults = query.execute();
                return this.makeAsciiResponse(queryResults);
}

function makeAsciiResponse(queryResults)
{
                var xmlResults = new Array();
                var response = new Array();
                for(i=0; i < queryResults.size(); i++)
                {
                                var toXml = new AsciiToXml(queryResults.get(i));
                                xmlResults[i] = toXml.execute();
                                var makeResponse = new MakeResponseAscii(queryResults.get(i), xmlResults[i]);
                                response[i] = makeResponse.execute();
                }

                return response;
}

MetarRequest.prototype.metarQuery = metarQuery;

MetarRequest.prototype.makeAsciiResponse = makeAsciiResponse;

// Code the user writes:
var dataRequest = new MetarRequest();
dataRequest.metarQuery(3);
```

JavaScript Scripting: Object Oriented

- JavaScript supports user-defined objects
  - An object is a container for attributes, which can be values, functions, or objects
  - This allows creation of class objects similar to those in OO languages
    - Internally, the class instance is referenced via the *this* keyword
- Syntax used for class object definition/creation is shown at right
- The class constructor is a function that is used to create the object
  - Use the *this* keyword to assign instance attributes
- The *prototype* object is used to create class attributes
  - All instances share the class attributes
  - Class methods are assigned to the *prototype* object

```
/* the class constructor */
function HelloWorld() {
  this.message = "";
}

/* class methods */
function _execute() {
  var logger = new SystemLog();
  logger.log("info",this.message);
  /* empty response */
  var response = new MakeResponseNull("Hello World.",
                                      new TermQuery("obs"));
  return response.execute();
}

function _setMessage(text) {
  this.message = text;
}

/* attach methods to class - w/aliases */
HelloWorld.prototype.execute = _execute;
HelloWorld.prototype.setMessage = _setMessage;

/* script to use the class */
var runner = new HelloWorld();
runner.setMessage("Hello World");
runner.execute();
```

JavaScript Scripting: Object Oriented (cont'd)

- Class definitions may be contained in a separate .js file
  - ADE 1.0 includes sample JavaScript definition files which are used by the Micro Engine Test Web pages
  - IN ADE 1.0, JavaScript class definitions are located in ~/opt/esb/js
- The client application uses an "include" statement to cause the µEngine script runner to load the class file.

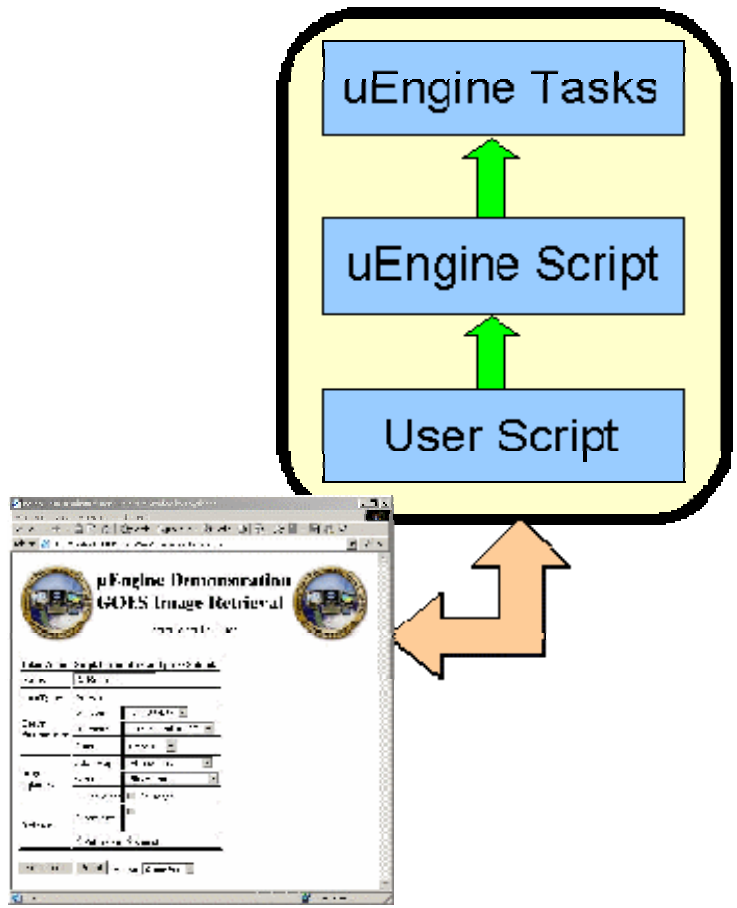
```
/* the class constructor */
function HelloWorld() {
  this.message = "";
}

/* class methods */
function _execute() {
  var logger = new SystemLog();
  logger.log("info",this.message);
  /* empty response */
  var response = new MakeResponseNull("Hello World.",
                                      new TermQuery("obs"));
  return response.execute();
}

function _setMessage(text) {
  this.message = text;
}

/* attach methods to class - w/aliases */
HelloWorld.prototype.execute = _execute;
HelloWorld.prototype.setMessage = _setMessage;
```

```
/* The client submits this script */
include("HelloWorld.js");
var runner = new HelloWorld();
runner.setMessage("Hello World");
runner.execute();
```

This approach allows for the creation of utility scripts with a simplified user interface.

JavaScript Scripting: Three-Tiered Approach

- Object approach outlined on previous slides allows for a three-level approach to µEngine development
  - µEngine tasks are created to perform a specific function such as querying the data store or performing math functions on sets of data
  - a µEngine is created to perform a general task such as retrieving satellite imagery. script is written as a JavaScript class with setter methods for key script parameters.
  - A user writes a short script to utilize the JS class. The actual code for creating the script may be hidden behind a GUI interface.
- The AWIPS Test Driver Web page uses this approach!



Programming Example

- Convert the sample GRIB → Image script (slide 82) into a JavaScript class and create the "user" script.

Hints:

- Provide:
  - a setter for the number of images
  - a setter for the query parameters
  - a setter for the color map
  - a setter for the image type

```
gribRequest();

function gribRequest()
{
  var plugin = "grib";
  var query = new TermQuery(plugin);
  query.setCount(3);
  query.addParameter("paramid", "Temperature");
  query.addParameter("levelinfo", "50000.0_Pa");
  var queryResults = query.execute();
  var responses = new Array();

  for(i=0; i < queryResults.size(); i++)
  {
    var currentQuery = queryResults.get(i);
    var fileIn = new FileIn(plugin, currentQuery);
    var gribMap = new GribMap(plugin, "GribRGB",
                              fileIn.execute(), currentQuery);
    var imageData = gribMap.execute();
    var colorMap = new ColorMapImage("GribRGB", imageData,
                                     gribMap.getGridGeometry());
    var format = "png";
    var imageOut = new ImageOut(colorMap.execute(), format,
                                gribMap.getGridGeometry());
    var fileOut = new FileOut(imageOut.execute(), format);
    var makeResponse = new MakeResponseUri(fileOut.execute(), null,
                                           currentQuery.getDataURI(), format);
    responses[i] = makeResponse.execute();
  }
  return responses;
}
```

Use Eclipse to demo the file creation – comments omitted.

Programming Example: The JavaScript

Execute Method

Constructor

Setters

Map Methods to Object Prototype

User Script

```
function GribRequest() {
  this.plugin = "grib";
  this.format = "png";
  this.colormap = "GribRGB";
  this.query = new TermQuery(this.plugin);
}

function _execute() {
  var queryResults = this.query.execute();
  var responses = new Array();

  for(i=0; i < queryResults.size(); i++)
  {
    var currentQuery = queryResults.get(i);
    var geom = currentQuery.getGrid().getGridGeom();
    var fileIn = new FileIn(this.plugin, currentQuery);
    var gribMap = new GribMap(this.plugin, this.colormap,
                              fileIn.execute(),geom);
    var imageData = gribMap.execute();
    var colorMap = new ColorMapImage(this.colormap, imageData,
                                     gribMap.getGridGeometry());
    var imageOut = new ImageOut(colorMap.execute(), this.format,
                                gribMap.getGridGeometry());
    var fileOut = new FileOut(imageOut.execute(), this.format);
    var makeResponse = new MakeResponseUri(fileOut.execute(), null,
                                           currentQuery.getDataURI(),
                                           this.format);
    responses[i] = makeResponse.execute();
  }
  return responses;
}
function _addParameter(name,value) {
  this.query.addParameter(name,value);
}
function _setCount(count) {
  this.query.setCount(count);
}
function _setColorMap(map) {
  this.colormap = map;
}
function _setImageType(type) {
  this.format = type;
}
GribRequest.prototype.execute = _execute;
GribRequest.prototype.addParameter = _addParameter;
GribRequest.prototype.setCount = _setCount;
GribRequest.prototype.setColorMap =  setColorMap;
GribRequest.prototype.setImageType = _setImageType;

var runner = new GribRequest();
runner.addParameter("paramid", "Temperature");
runner.addParameter("levelinfo", "50000.0_Pa");
runner.setCount(3);
runner.setColorMap("GribRGB");
runner.setImageType("png");
runner.execute();
```

Client Applications

- Clients can build Object Oriented JavaScript and include it in the script behind the scenes – user has to write very little code to run scripts
- Clients can extend uEngine functionality by adding JavaScript methods
  - Example: Calculate wind chill off MetarRecord returned by a TermQuery
- CAVE could use Eclipse plug-in architecture to embed a JavaScript IDE and assist users in creating scripts

Client Applications (cont'd)

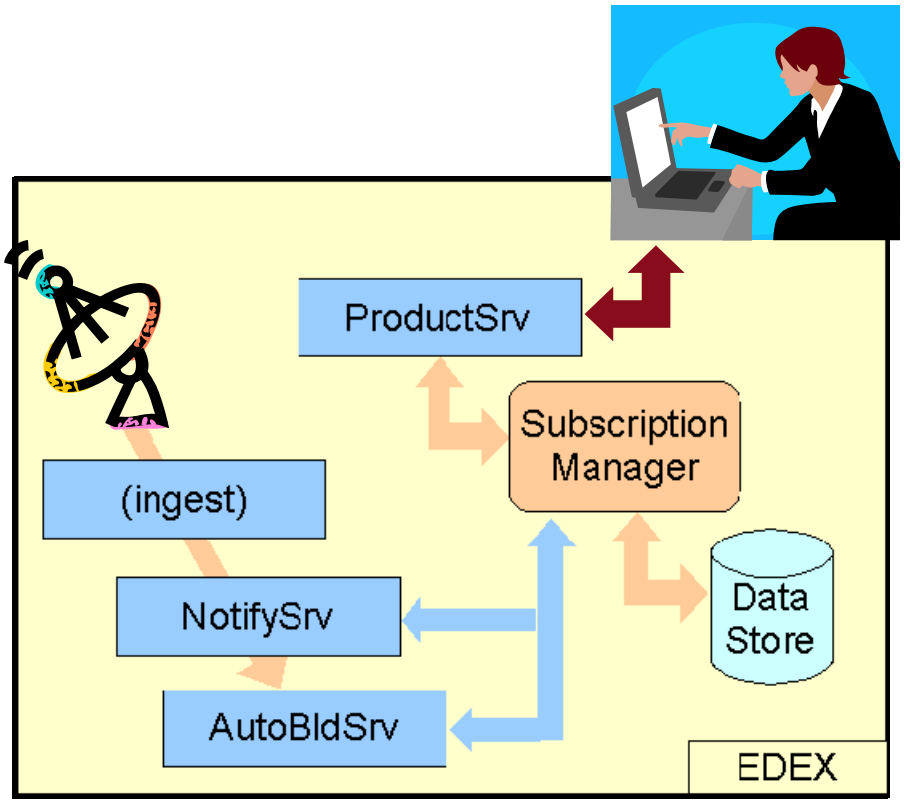
Example Query Using the METAR Script:

A user only needs to write three lines if the library is imported or if the client application includes it.

```
var metarData = new MetarData();
metarData.displayWindChill(true);
metarData.metarQuery();
```

JavaScript Scripting: Subscriptions

- EDEX supports subscribing for a product
  - The subscription is based on the data type and a script used to create the product
  - CAVE uses subscription to create auto updating loops
- When data arrives, EDEX replaces the general query originally used with a specific data item
- This requires specially constructed scripts
  - Scripts are limited to a single data request
  - The script must be *subscription enabled*
  - Subscription does not work reliably (as of ADE 1.0) in a clustered environment



JavaScript Scripting: Subscriptions (cont'd)

- To make a script subscribable:
  - Script must be written as a JavaScript class
  - TermQuery task query must be an instance variable
  - TermQuery task's three-argument constructor must be used
    - Arguments must be as shown
  - Class must have a *subscription* instance variable; this will contain a *Subscription* object
  - Class' main action method (usually _execute) must set up the subscription object

```
/* class constructor */
function ProductSubscribe() {
  this.plugin = "radar";
  /* other class attributes as needed */

  /* subscription attributes */
  this.subscribe = false;
  this.subscription = null;
  /* the query object */
  this.query = new TermQuery(this.plugin,
                             subscriptionDataFieldId,
                             subscriptionDataQueryId);
}
/* main action method */
function _execute() {
  /* setup the subscription */
  if(this.subscribe){
    this.subscription = new Subscription();
    this.subscription.setup(this.query);
  }
  /* additional action as needed */
}
/* subscription switch */
function _enableSubscription(){
  this.subscribe = true;
}
/* setter methods as needed */
/* map the methods into the class prototype */
ProductSubscribe.prototype.execute = _execute;
ProductSubscribe.prototype.enableSubscription =
_enableSubscription;
/* additional mappings as needed */
/* execute the script */
var runner = new ProductSubscribe();
/* set attributes as needed */
/* set the subscription */
runner.enableSubscription(true);
runner.execute();
```

Programming Example

- Convert the sample GRIB → Image script (slide 82) into a subscription script.

```
gribRequest();

function gribRequest()
{
  var plugin = "grib";
  var query = new TermQuery(plugin);
  query.setCount(3);
  query.addParameter("paramid", "Temperature");
  query.addParameter("levelinfo", "50000.0_Pa");
  var queryResults = query.execute();
  var responses = new Array();

  for(i=0; i < queryResults.size(); i++)
  {
    var currentQuery = queryResults.get(i);
    var fileIn = new FileIn(plugin, currentQuery);
    var gribMap = new GribMap(plugin, "GribRGB",
                              fileIn.execute(), currentQuery);
    var imageData = gribMap.execute();
    var colorMap = new ColorMapImage("GribRGB", imageData,
                                     gribMap.getGridGeometry());
    var format = "png";
    var imageOut = new ImageOut(colorMap.execute(), format,
                                gribMap.getGridGeometry());
    var fileOut = new FileOut(imageOut.execute(), format);
    var makeResponse = new MakeResponseUri(fileOut.execute(), null,
                                           currentQuery.getDataURI(), format);
    responses[i] = makeResponse.execute();
  }
  return responses;
}
```

Use Eclipse to examine the code.

Programming Example: Grib Request Class for Subscription

Raytheon

```
function GribRequest() {
  this.plugin = "grib";
  this.format = "png";
  this.colormap = "GribRGB";
  this.subscribe = false;
  this.subscription = null;
  this.query = new TermQuery(this.plugin,
                             subscriptionDataFieldId,
                             subscriptionDataQueryId);
}

function _execute() {
  if (this.subscribe) {
    this.subscription = new Subscription();
    this.subscription.setup(this.query);
  }
  var queryResults = this.query.execute();
  var responses = new Array();

  for(i=0; i < queryResults.size(); i++)
  {
    var currentQuery = queryResults.get(i);
    var geom = currentQuery.getGrid().getGridGeom();
    var fileIn = new FileIn(this.plugin, currentQuery);
    var gribMap = new GribMap(this.plugin, this.colormap,
                              fileIn.execute(),geom);
    var imageData = gribMap.execute();
    var colorMap = new ColorMapImage(this.colormap, imageData,
                                     gribMap.getGridGeometry());
    var imageOut = new ImageOut(colorMap.execute(), this.format,
                                gribMap.getGridGeometry());
    var fileOut = new FileOut(imageOut.execute(), this.format);
    var makeResponse = new MakeResponseUri(fileOut.execute(), null,
                                           currentQuery.getDataURI(),
                                           this.format);
    responses[i] = makeResponse.execute();
  }
  return responses;
}
function _addParameter(name,value) {
  this.query.addParameter(name,value);
}
function _setCount(count) {
  this.query.setCount(count);
}
function _setColorMap(map) {
  this.colormap = map;
}
function _setImageType(type) {
  this.format = type;
}
function _enableSubscription() {
  this.subscribe = true;
}
GribRequest.prototype.execute = _execute;
GribRequest.prototype.addParameter = _addParameter;
GribRequest.prototype.setCount = _setCount;
GribRequest.prototype.setColorMap = _setColorMap;
GribRequest.prototype.setImageType = _setImageType;
GribRequest.prototype.enableSubscription = _enableSubscription;

var runner = new GribRequest();
runner.addParameter("paramid", "Temperature");
runner.addParameter("levelinfo", "50000.0_Pa");
runner.setCount(3);
runner.setColorMap("GribRGB");
runner.setImageType("png");
runner.enableSubscription();
runner.execute();
```

Code modifications are in **bold**.

Micro Engine: Known Issues

- µEngine cannot use “C++” style comments (corrected – TO8)
- Subscription has some problems, generally related to clustering
- Radar GeoTIFF retrievals are broken
- The Spatial Query task is broken and should not be used

Summary

- Architecture includes .js libraries
  - Supports subscriptions
- Simpler tasks
  - POJOs with an execute() method
  - Eliminates chain data, metadata, and digester rules
  - Less code
- Advanced scripts
  - Object Oriented potential
  - Flexible customization
  - Client applications can extend functionality of uEngine

Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 4: Data Type Plug-in (rev. 1)

February 14, 2008

AWP.TRG.SWCTR/TO6.ADE/CAVE04.01

Objectives

- Learn why the Plug-In pattern was chosen
- Understand the architectural pattern of a Data Type Plug-In
- Write a new Plug-In for a new data type
- Put a MicroEngine task into a Data Type Plug-In

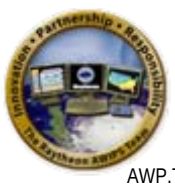
Estimated Time: 3 hours

Notes:

- *Some slides reiterate material previously presented, and are included here for completeness. Others provide additional information for the developer. These slides will not be covered in this briefing.*
- *Questions? Please ask at any time.*

Prerequisites

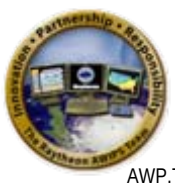
- Access to an installation of ADE 1.0
- Familiarity with utilizing Eclipse for Java development
- Familiarity with Object Oriented Programming
- Programming experience in the Java programming language
  - ADE 1.0 utilizes Java 1.6



Topics

- Data Type Plug-In Concepts
- Plug-In Archive/Build Details
- AWIPS EDEX Plug-In Architecture
- Data Record Objects
- Data Plug-In Configuration
- ADE Plug-in Creation Tool
- Adding Plug-In to EDEX Ingest
- Adding uEngine Data-Type Processing

Data Type Plug-In Concepts



Why a Plug-In Pattern?

- Must be able to:
  - Add new data processing capability with minimal impact to the remaining system
  - Extend a deployed system to new data sources
  - Fully integrate new data sources into the system so that all the standard display and analysis features can be used
  - Add custom data transforms to a deployed system
  - “Hot deploy” to add new data types to the existing system

What Is a Plug-In in Java Terms?

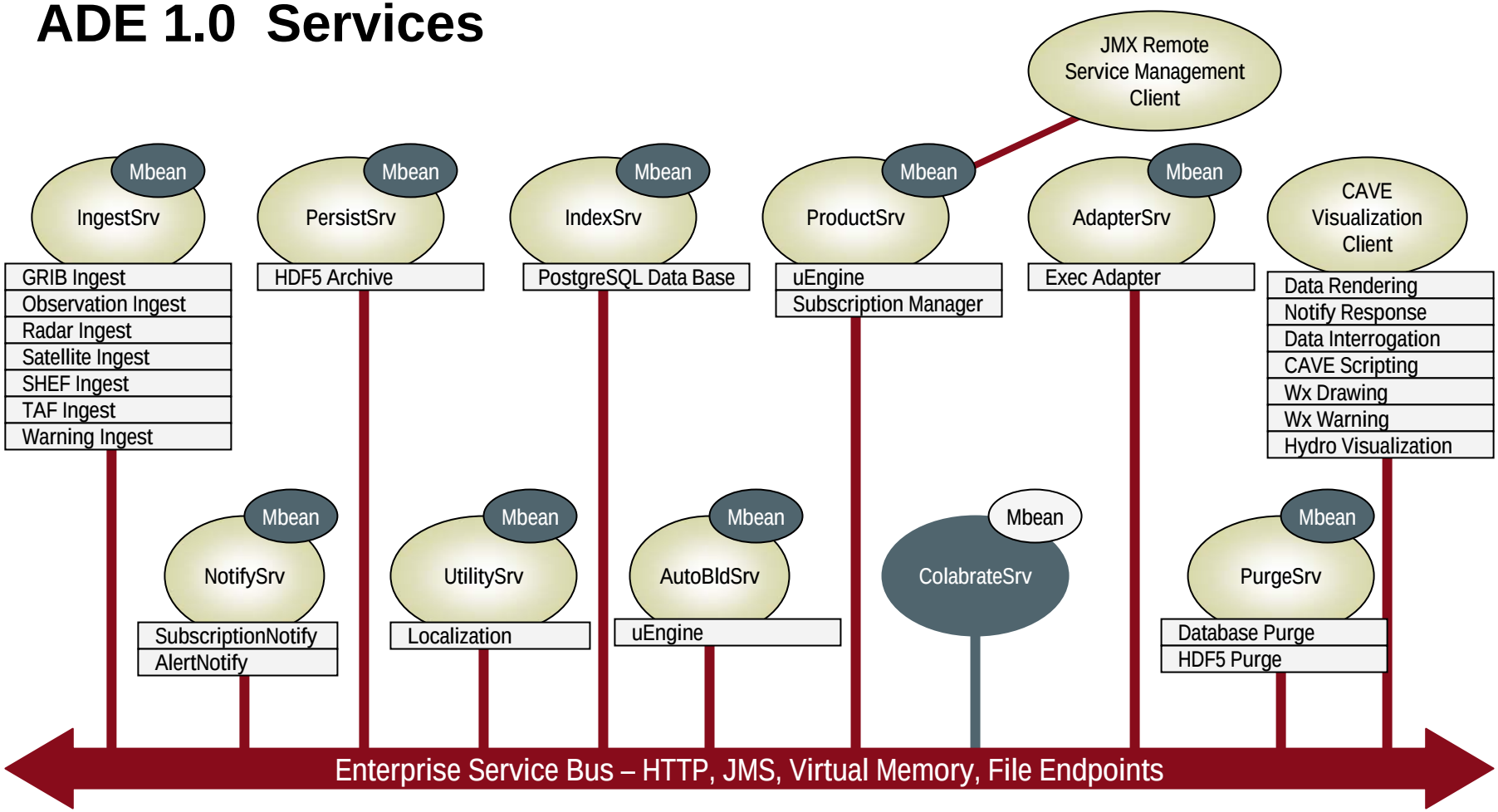
- Plug-In:
  - Encapsulates all processing needed for a data type
  - Implements classes exposed through Java interfaces
  - Classes accessed by interface, and data type through a plug-in factory
- Plug-In Factory:
  - Uses Java class loading to create the appropriate class for the requested interface
  - Is XML configurable
- No change to existing code when new Plug-In is added
- All data ingest completed through a Plug-In
  - ASCII data processing in the MicroEngine: All Plug-In based!

Easy Adaptability to New Data Types

- **Plug-In Vision:** Create a component that can be plugged into a deployed system to adapt the system to a new data type. Plug-In will enable ingest, storage, retrieval, and transformation of data.
- **Plug-In Implementation:** Java plug-in pattern with a class loader to allow dynamic deployment. Metadata extraction and indexing are based on a metadata store with a URI-based data repository.
  - ***Note:*** *Plug-ins become possible with Java's "Interface" class and Java's concept of class loading.*

Plug-In Interface to SOA Services

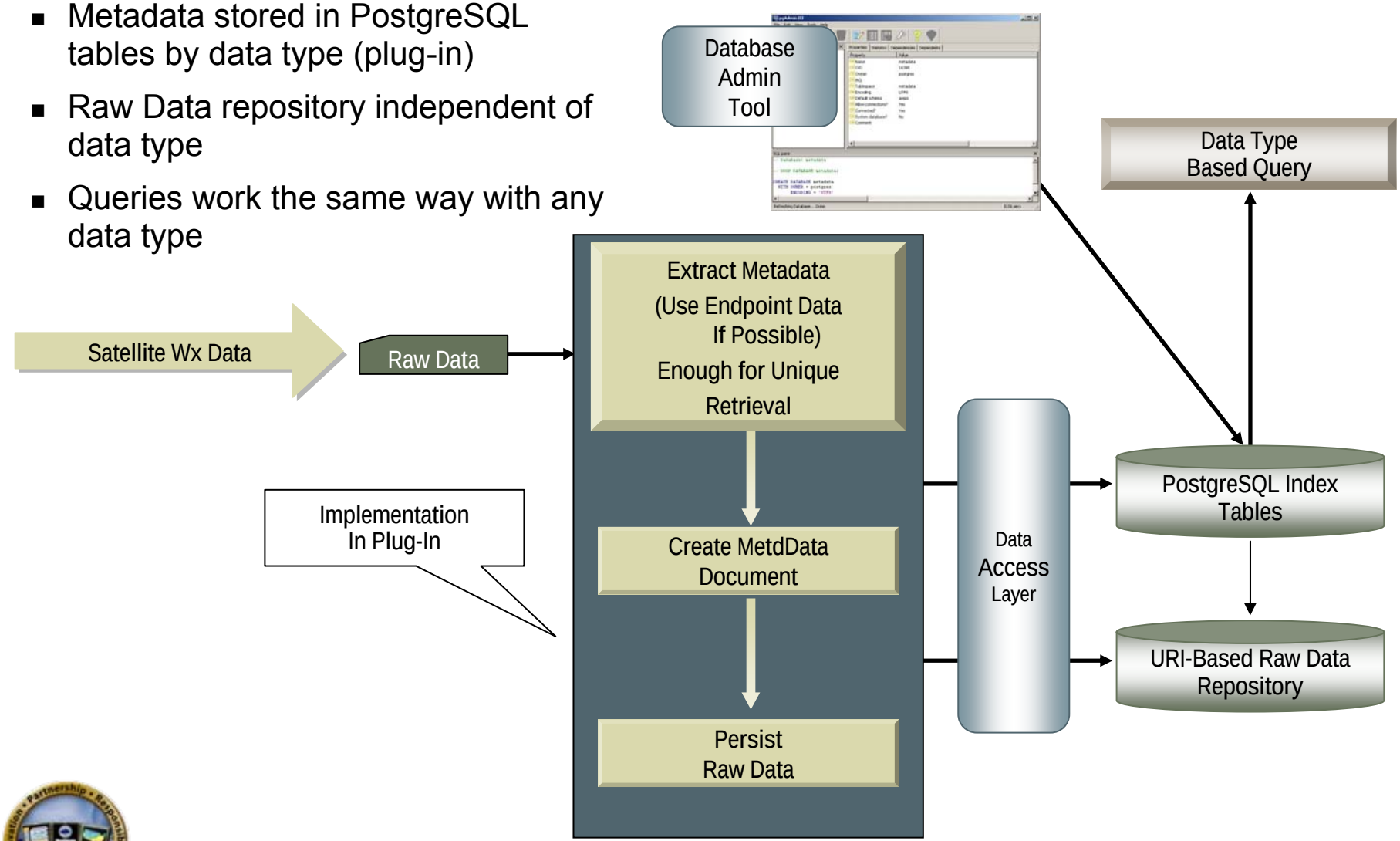
ADE 1.0 Services



Services Independent of End Points

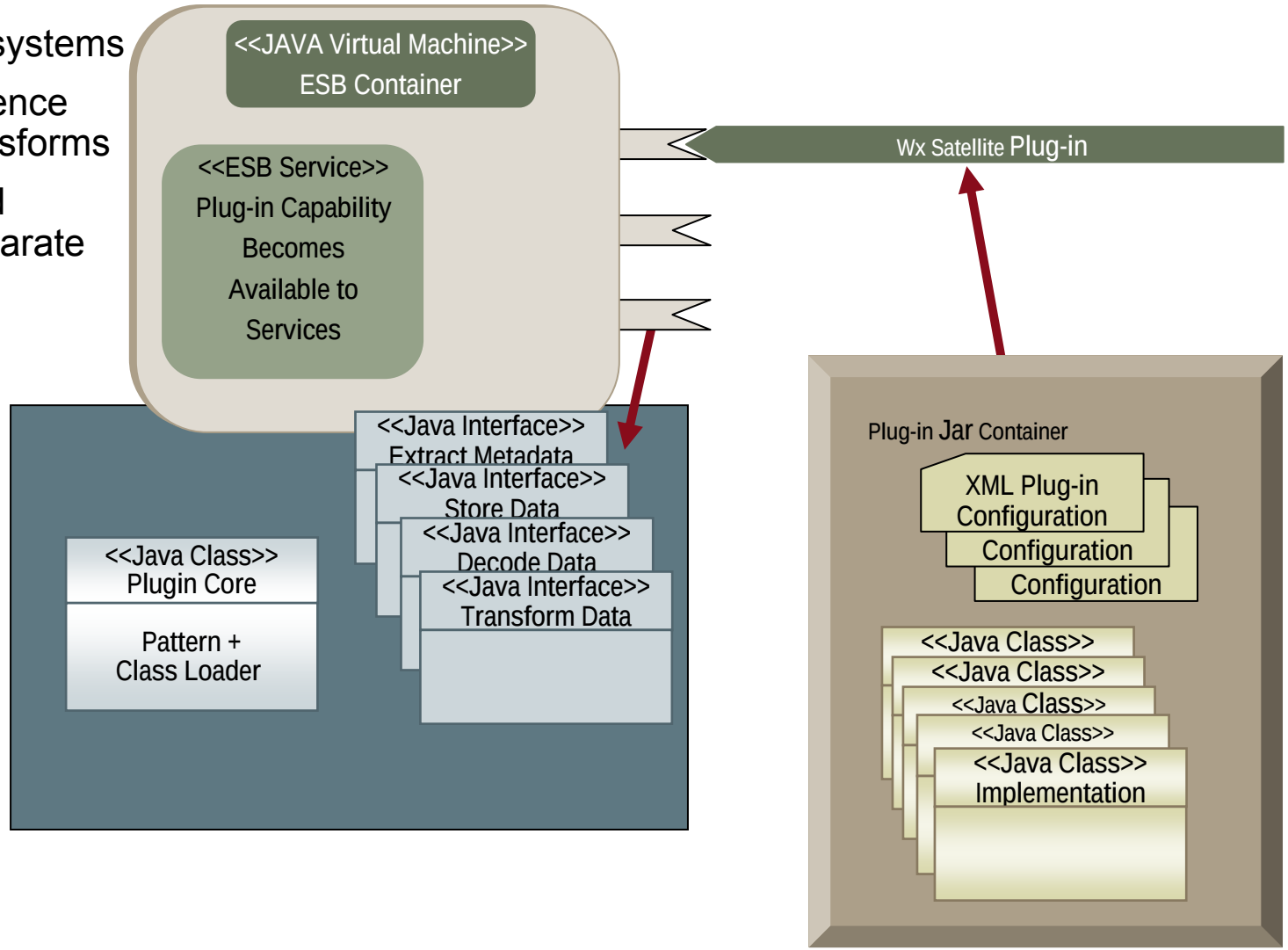
High-Level Concept Diagram

- Metadata stored in PostgreSQL tables by data type (plug-in)
- Raw Data repository independent of data type
- Queries work the same way with any data type



Java Files Involved in Plug-In Concept

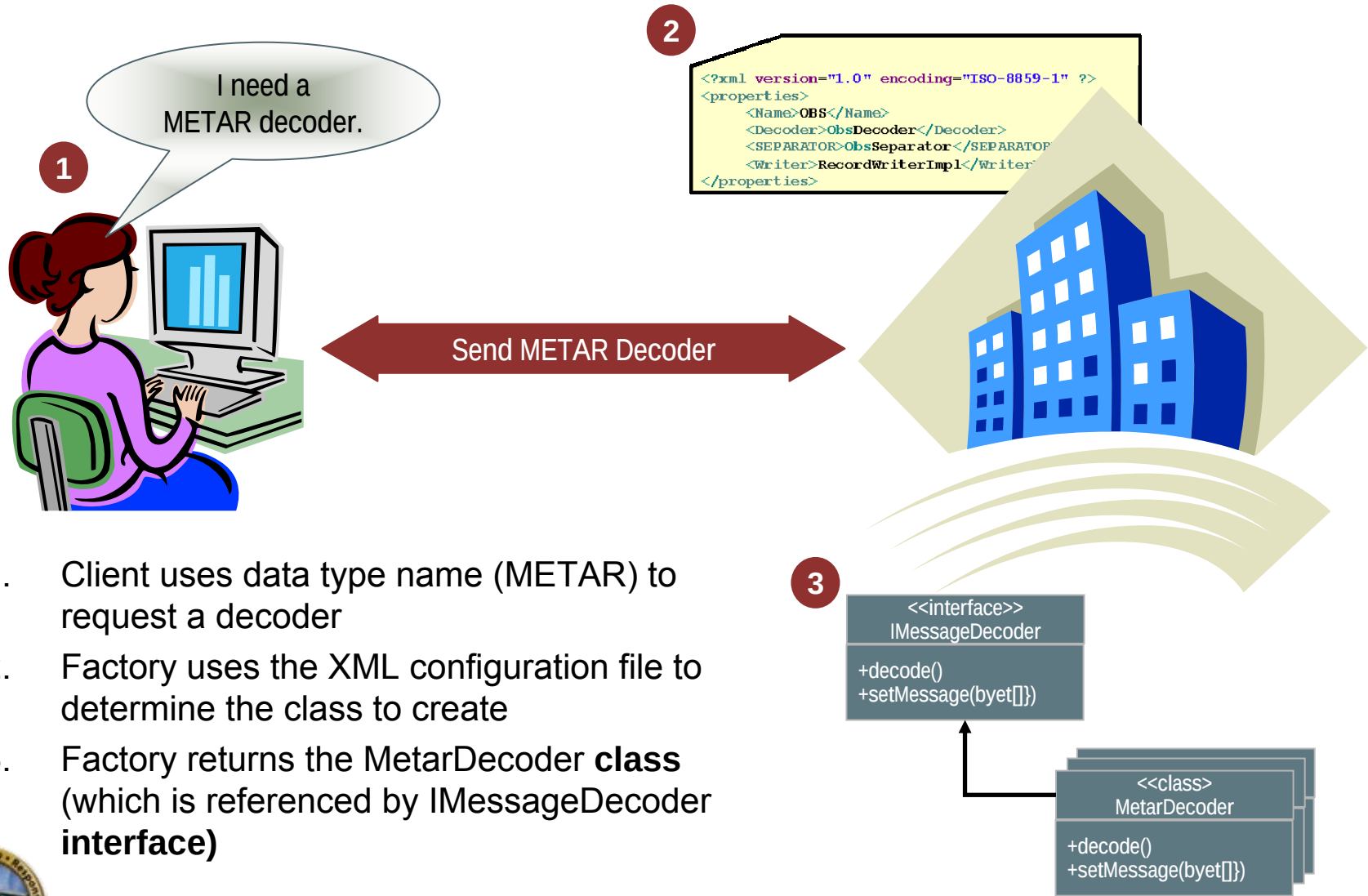
- Enables new / modified data types to be deployed to live systems
- Enables new science through new transforms
- Plug-Ins built and packaged as separate components



Data Type Plug-In Factory Concept

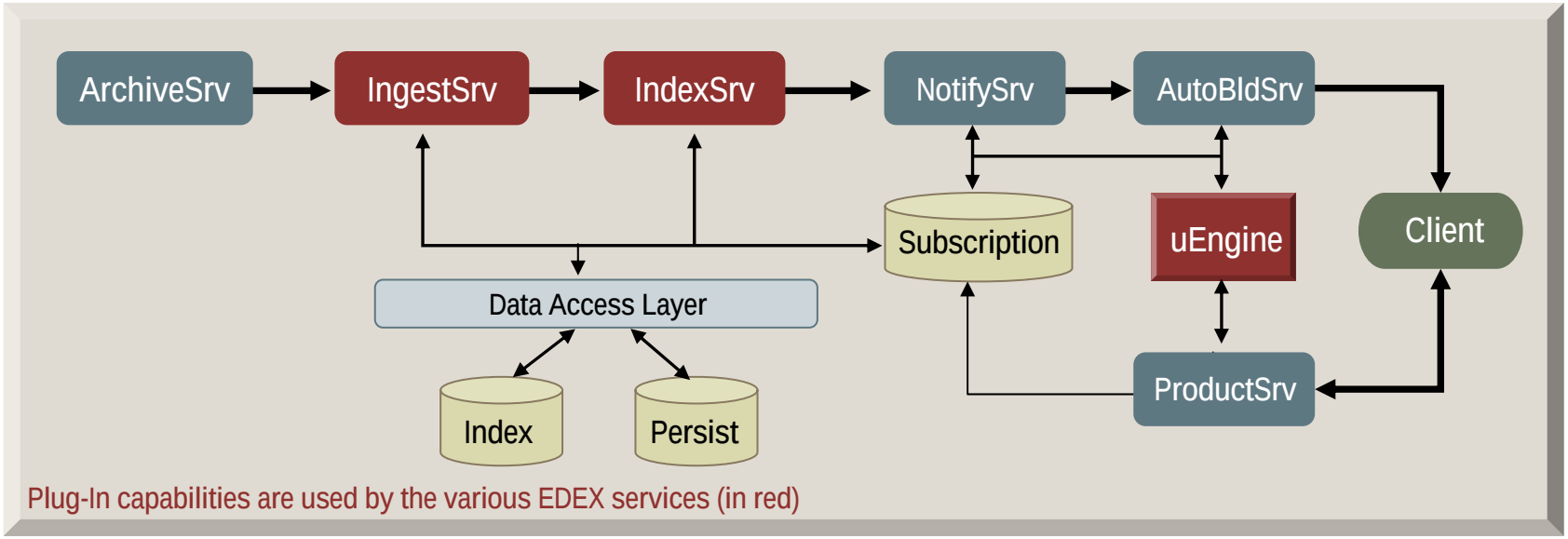
- **The Vision:** Data type Plug-In provides the means for ingesting data into the system and exporting data from the system. Internally (to the extent possible), this should be handled in a type-neutral format.
- **The Implementation:** Goal of the data type Plug-In definition is to encapsulate the data handling of a data capabilities into a package having well-defined interfaces.
- **The Factory:** In OO, a “factory” is defined as a class that creates and delivers instances of a class implementing a specific interface. Client requests the instance using metadata, for example, the name of the Plug-In.

Factory Concept



1. Client uses data type name (METAR) to request a decoder
2. Factory uses the XML configuration file to determine the class to create
3. Factory returns the MetarDecoder **class** (which is referenced by IMessageDecoder **interface)**

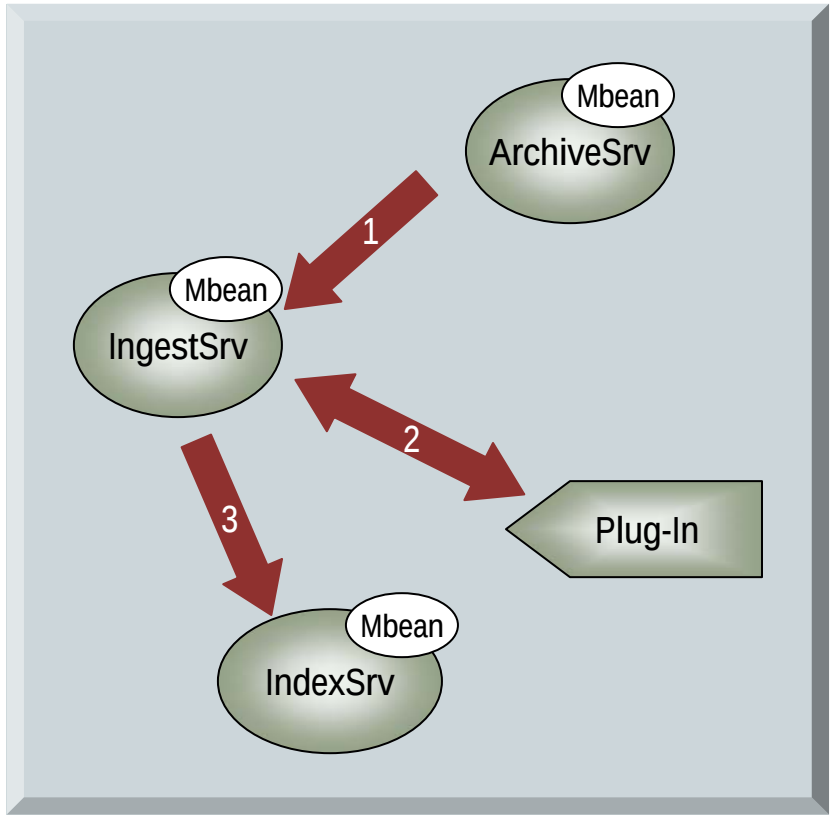
AWIPS EDEX Data Flow & Plug-Ins



- Data type Plug-In used for processing as data flows from Ingest Server to Client
- Plug-In architecture allows implementation of generic (data neutral) services
  - ***Notes:***
    - *Except for Ingest, a single service handles all data types.*
    - *Ingest uses a separate service for each data type, but all use the same Java class.*
- Plug-In architecture supports data flexibility

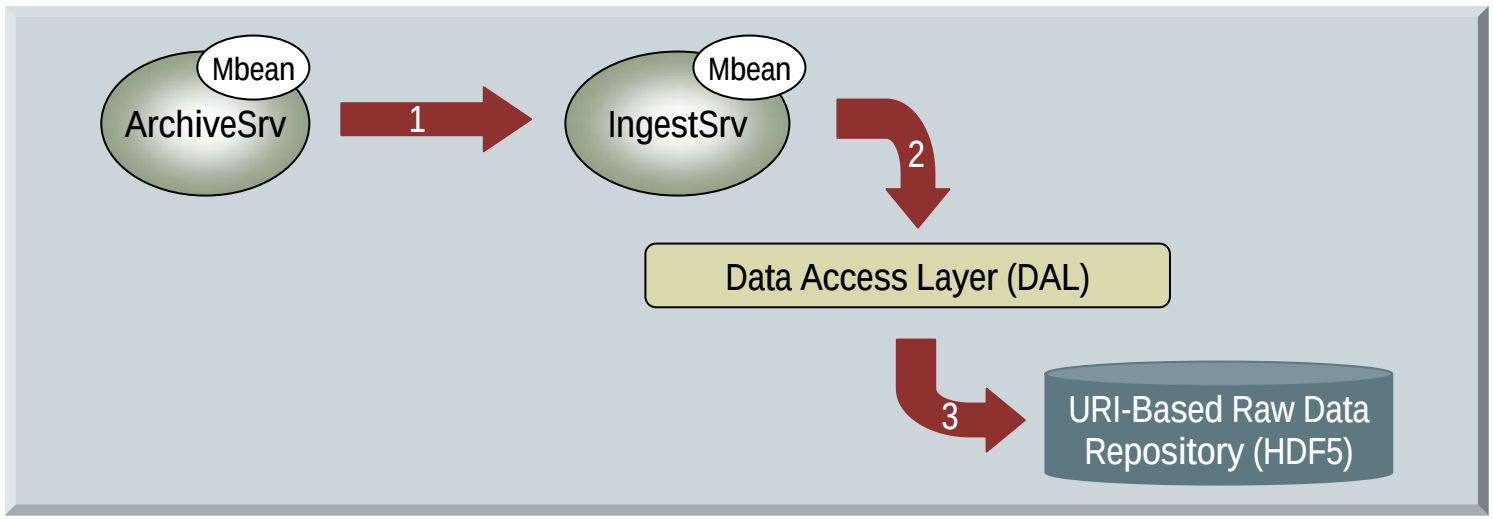
Plug-In Relationship to Metadata

- Each Ingest Service end point works with a specific type of data
  - End point receives a file pointer from ArchiveSrv via Java Message Service (JMS) message
  - IngestSrv uses information obtained from end point configuration to request appropriate Plug-In to process the file
  - Plug-In decodes the metadata, then passes the metadata (including Plug-In name) to the Index Service



Plug-In Relationship to Data Persistence

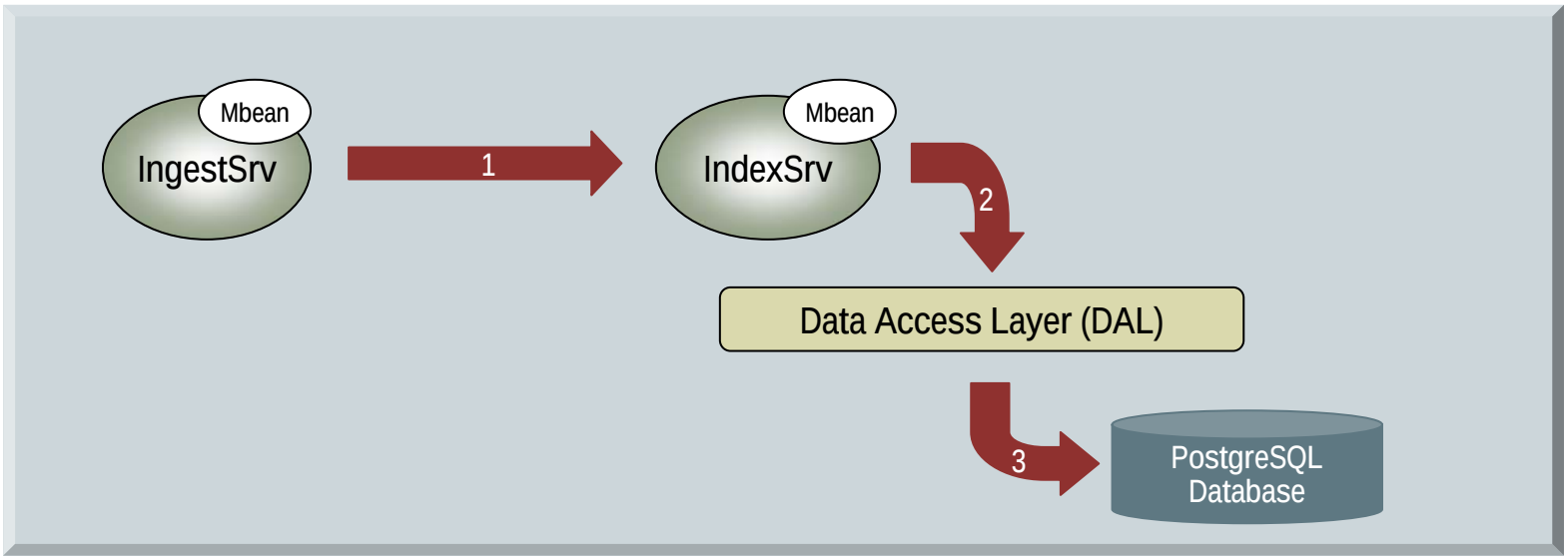
- For Plug-Ins that require separate data persistence, the data record implements the IPersistable interface.
  - Plug-In responsible for converting raw data into a format ready for persistence
  - For multi-record files, this includes separation of file into records
- For persistable data, the Ingest Server end point outputs the “Ready to Store” data to the data store via the Data Access Layer (DAL)



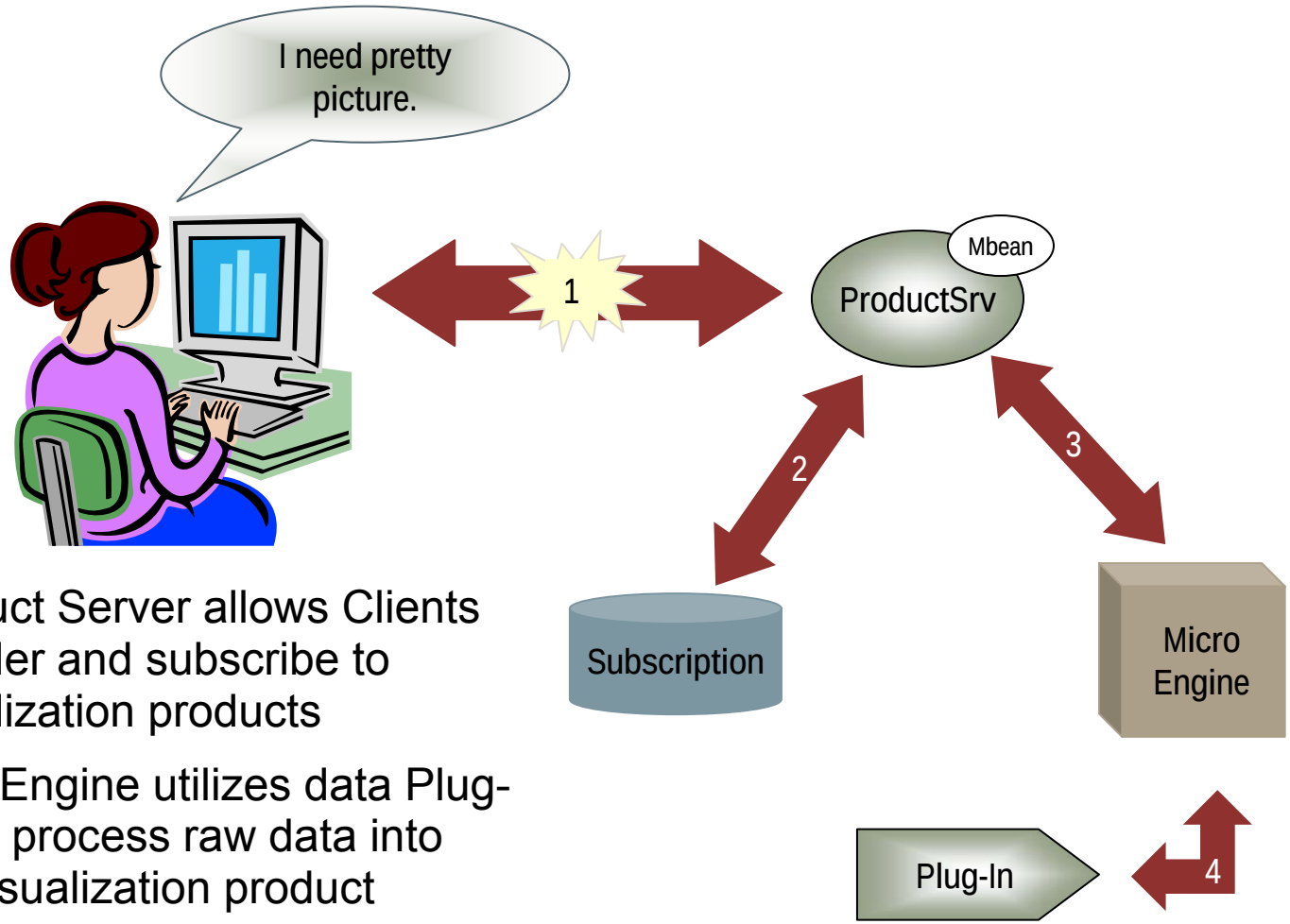
Note: Most of the plug-in work is done in the Ingest Server.

Plug-In Relationship to Data Indexing

- Plug-In responsible for extracting the Metadata from the data
- Each Ingest Server endpoint passes the Metadata to the Index Server via the Persist Server



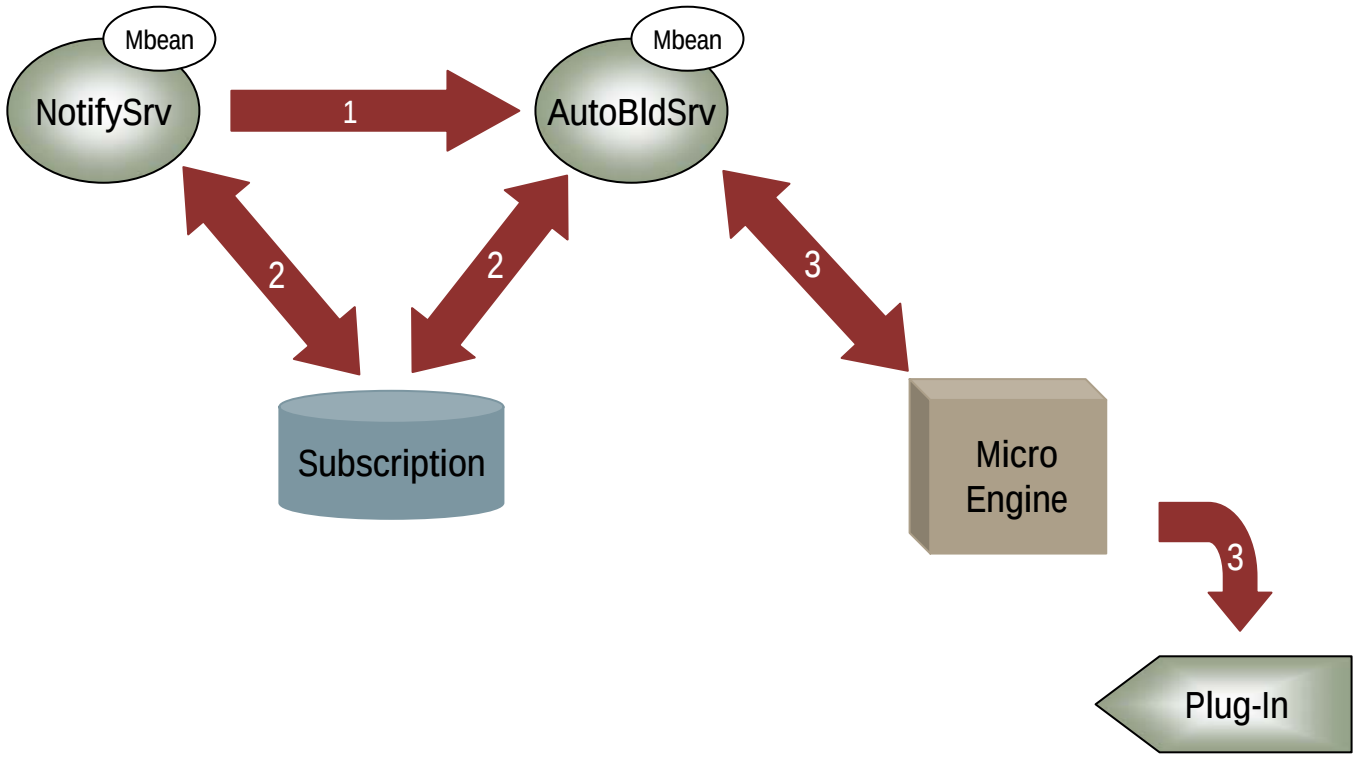
Plug-In Relationship to Product Server



- Product Server allows Clients to order and subscribe to visualization products
- MicroEngine utilizes data Plug-Ins to process raw data into the visualization product

Plug-In Relationship to Notification

- Notification triggers automatic building of visualization products
- Auto-build process utilizes Plug-Ins for processing

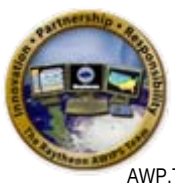


Note: *Plug-In utilization is mainly in MicroEngine.*

BREAK


Innovation • Partnership • Responsibility
The Raytheon Mark's Team

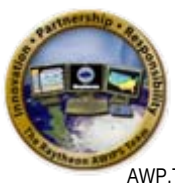
Plug-In Archive/Build Details



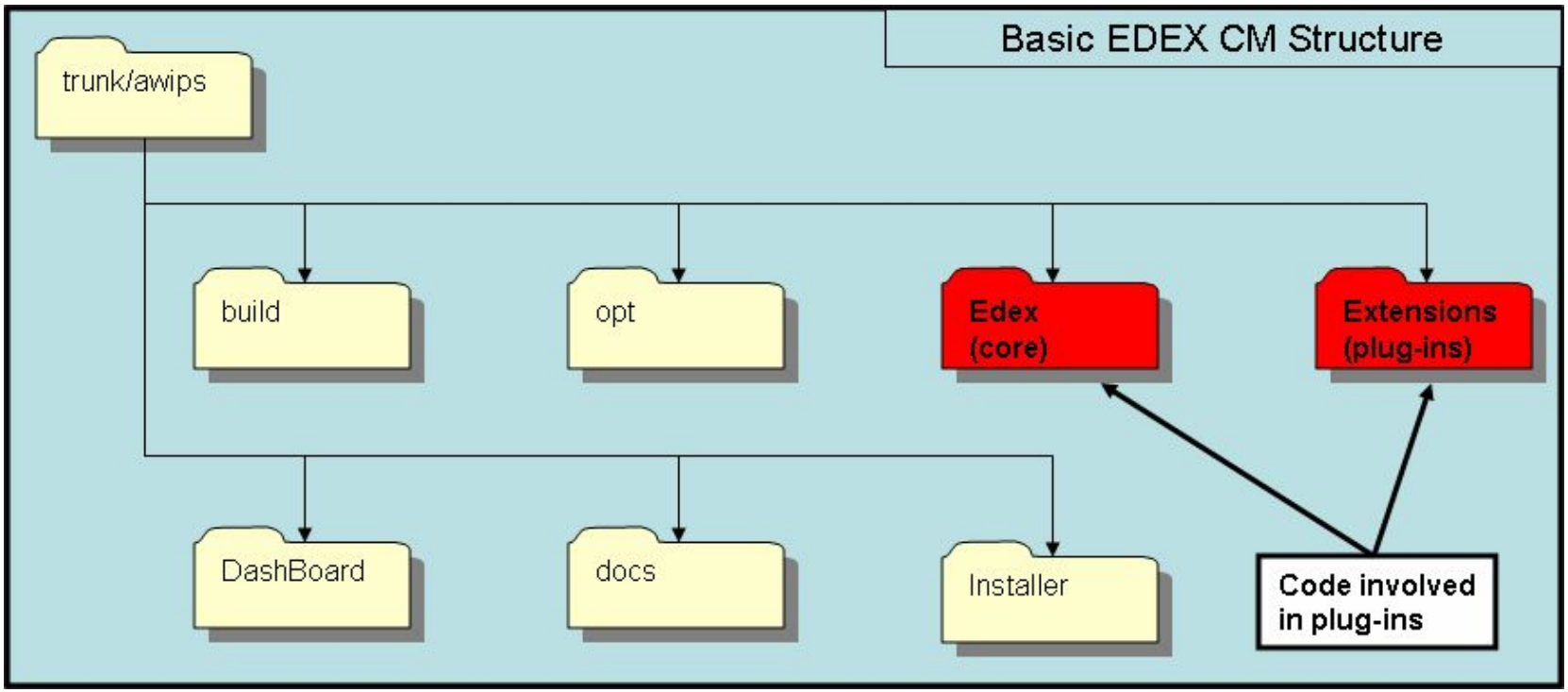
AWIPS EDEX Project Organization

About This Section

- Examines the organization of the ADE's EDEX code base, which doubles as a build structure
- Outlines a Plug-in for decoding "message" data – as examples for this section and code examples for the following sections
  - ***Notes:***
    - *A "message" data record is a plain ascii text record consisting of a WMO header and a record body, while "message" decoding consists of separating the header and body and parsing the WMO header.*
    - *This plug-in illustrates how to create, build, and configure a new data-type plug-in for EDEX.*



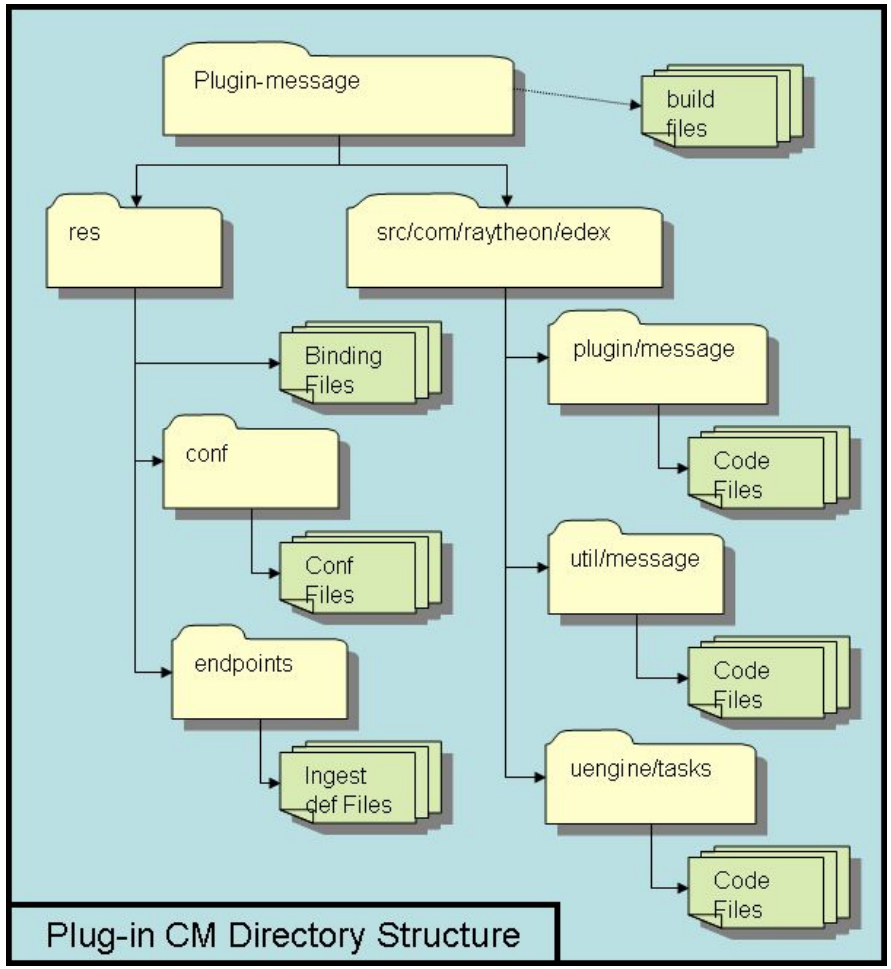
AWIPS EDEX Project Organization (cont'd)



- ADE EDEX baseline separated into folders to support manual and/or automated builds
- Plug-In code location: Under the Extensions directory. Core Plug-In support: Under EDEX directory

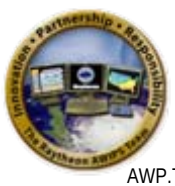
AWIPS Plug-In Directory Structure

- Each Plug-In contained in a directory structure under “trunk/awips/extensions”
- Two main branches in the Plug-In code:
  - src/… contains the code of the Plug-In, any supporting classes, and any uEngine tasks.
  - res/… contains configuration and other build-related files
- Directory structure under “src” defines the package structure for the Plug-In
  - All current Plug-Ins located in “com.raytheon.edex” package tree
  - Additional directories added as needed



AWIPS Plug-In Directory Structure Considerations

- Names of Plug-Ins
  - Unique name for each Plug-In
  - Always start with “plugin-”
  - Name for the “message” Plug-In: “plugin-message”



AWIPS Plug-In Directory Structure Considerations (cont'd)

Raytheon

- Three build-related files in each Plug-In:
  - build-component.properties
    - Located in the main Plug-In directory
    - Contains the build dependencies
    - Plug-Ins normally dependent on "common" and "uEngine"
  - client-includes.dat
    - Located in the main Plug-In directory
    - Identifies classes to include in the "client" version of the Plug-In
  - binding.xml
    - Located in the "res" directory of the Plug-In
    - Required only if the Plug-In defines any files that require JiBX-based serialization

AWIPS Plug-In Directory Structure

- First step in creating a Plug-In: Create the basic directory and build structure
  - Using Eclipse (or a file system browser), add basic directory structure for Plug-In under “trunk/awips/extensions”
  - Note Plug-In name, “message,” at the lowest level of the directory structure
  - This structure is created using the Plug-In creation tool
- Next: Create basic build file, build-component.properties
  - In most cases, the required dependencies are “common” and “uEngine” (as shown)

```
plugin-message
|-- res
|   |-- conf
|   `-- endpoints
`-- src
    `-- com
        `-- raytheon
            `-- edex
                |-- plugin
                |   `-- message
                `-- util
                    `-- message
```

build-component.properties:

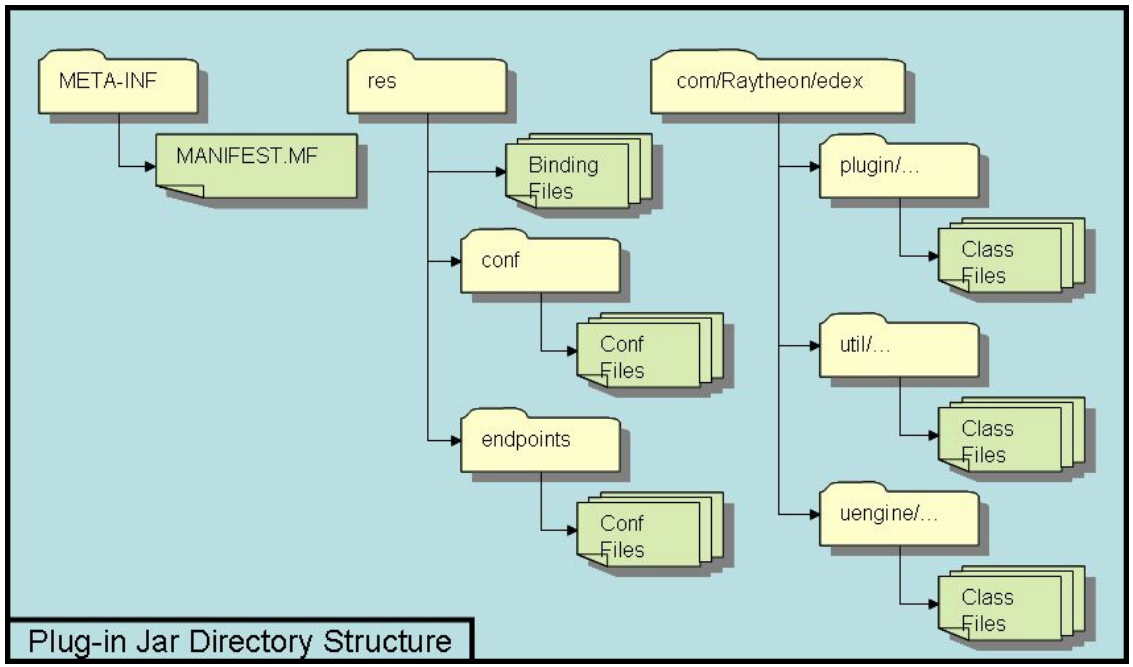
```
dependencies=common,uEngine
```

AWIPS EDEX Build Files to Modify

- Modify these build files when adding a new data type Plug-In:
  - trunk/awips/build/deployments/deployment.properties – add the Plug-In's main code directory to the deployment list
  - for the "message" Plug-In, this is plugin-message

Note: *This topic will be revisited later.*

AWIP EDEX Plug-In Jar Structure

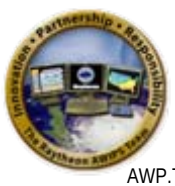


- Each JAR named to match the Plug-In directory in CM
  - CM location for the current METAR Plug-In: "trunk/awips/extensions/plugin-metar"
  - The resulting jar: "plugin-metar.jar"
- META-INF directory and MANIFEST.MF generated as part of the build process

BREAK


Innovation • Partnership • Responsibility
The Raytheon Team's Team

AWIPS EDEX Plug-In Architecture



Makeup of the Plug-In Architecture

- The Plug-in Decoder Proxy, which provides run-time access to the various data-type plug-ins
- An (optional) Data Access Object, which provides for storage of data in the AWIPS Persistence Store
- The Java classes needed to perform the tasks of the Plug-In
- A Java class providing a data object for each record of the data type
- Build files that control creation of the Plug-In
- Configuration files that define the operation of the Plug-In

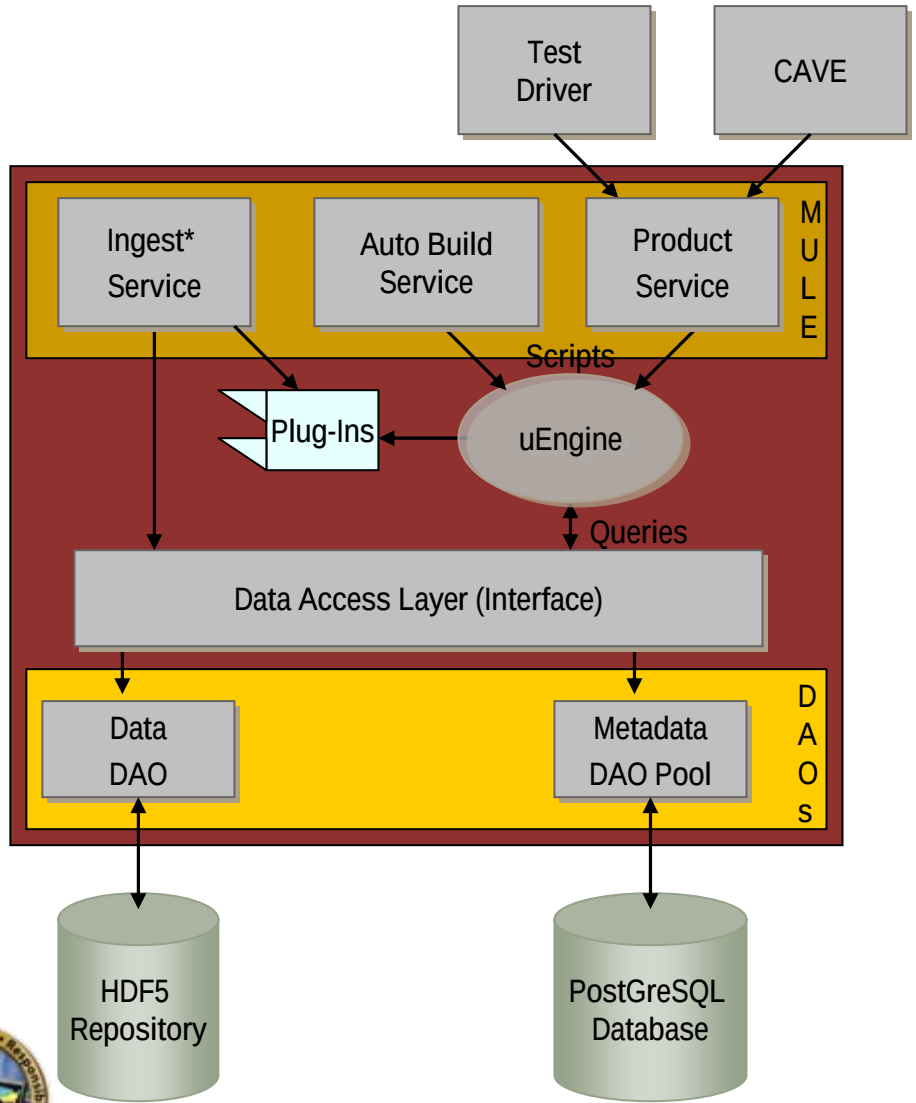
Plug-In Architecture: Data Record Classes

Raytheon

- Each data-type record represented internally by a Java class object called a “Data Record” object
  - Data Records extend the PluginDataObject abstract class (which is part of com.raytheon.edex.db.objects package)
    - PluginDataObject provides the minimal fields required for EDEX data persistence
    - Abstract classes in com.raytheon.edex.db.objects package also exist for other common data objects
- Data Record object created by the data-type decoder and used by the Data Access Layer (DAL) for metadata and data persistence
  - DAL provides a layer of abstraction above the data persistence mechanism that hides the actual implementation from the Plug-In

Note: *More on Data Record objects in the next section.*

Plug-In Architecture: Data Access Layer Design



This diagram shows the multi-tier design approach for AWIPS EDEX.

The test driver and CAVE communicate with the Data Access Layer (DAL) via the Product Server and the uEngine.

The DAL provides an interface to the business layer to decouple data storage implementation from business logic. This interface receives update and query requests. The DAL then delegates appropriate Data Access Objects (DAO) to interact with the data sources. The results are organized and returned to the business layer via the data layer interface.

This design enables the user to swap data sources without affecting the business layer, although a new DAO is necessary if the persistence method is changed.

Plug-In Architecture: The Plug-in Decoder Proxy Class

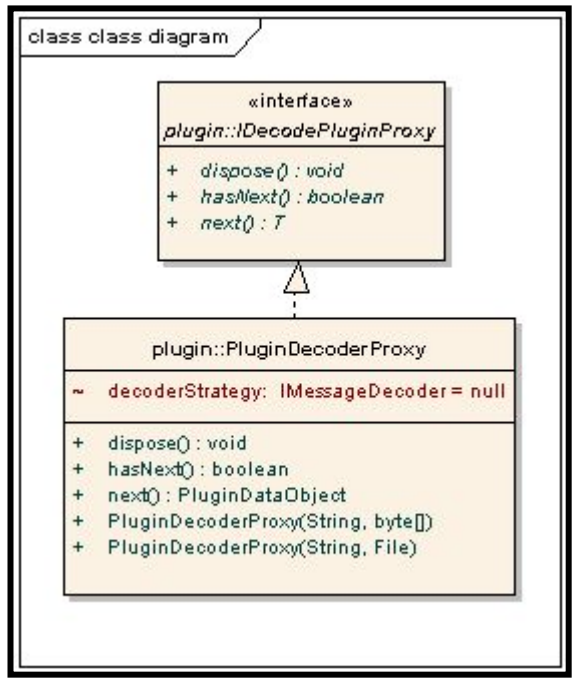
Raytheon

- Plug-In Decoder Proxy class provides interface between the Ingest Service and the data type decoders
  - Ingest Server creates an instance of the Plug-In Decoder Proxy class
    - Plug-In Decoder Proxy class uses the decoder factory to obtain additional classes to perform specific tasks, such as file decoding
  - Ingest Server uses Plug-In Decoder Proxy class methods to perform data operations
- Data type-specific functionality provided in two ways:
  - Implementing Plug-In interfaces
  - Configuring the DAL

Note: *More on both data type-specific functionality concepts later.*

AWIPS EDEX Plug-In Decoder Proxy

- Plug-In Decoder Proxy provides a configurable interface by which an EDEX Service may obtain the decoder for a data type
- Proxy configuration for a specific data-type plug-in contained in “plugin.xml,” which is found in the Plug-In’s “res/conf” directory
  - plugin.xml read by the EDEX Configuration Factory
  - Provides the information that allows the Plug-In Decoder Proxy to deliver the correct class(es) for a specific data type

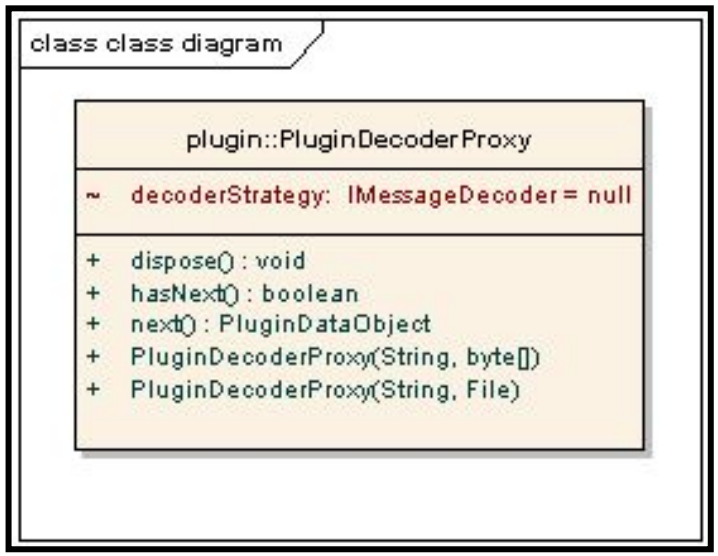



```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<properties>
     <Decoder>com.raytheon.edex.message.MessageDecoder</Decoder>
</properties>
```

Plug-In Architecture: Plug-In Decoder Proxy Class Design

Raytheon

- Methods available in the Plugin class:
  - PluginDecoderProxy() – (constructor) instantiates the class and obtains supporting class references from the Plug-In factory
  - hasNext() – determines if the data provided to the constructor has a data record to decode
  - next() – gets the next available (decoded) data record
  - dispose() – cleans up objects contained in the proxy – must be called before the class goes out of scope



Note: *Method arguments and return values are as specified in the class diagram.*

Plug-In Architecture: Plug-In Decoder Proxy Utilization

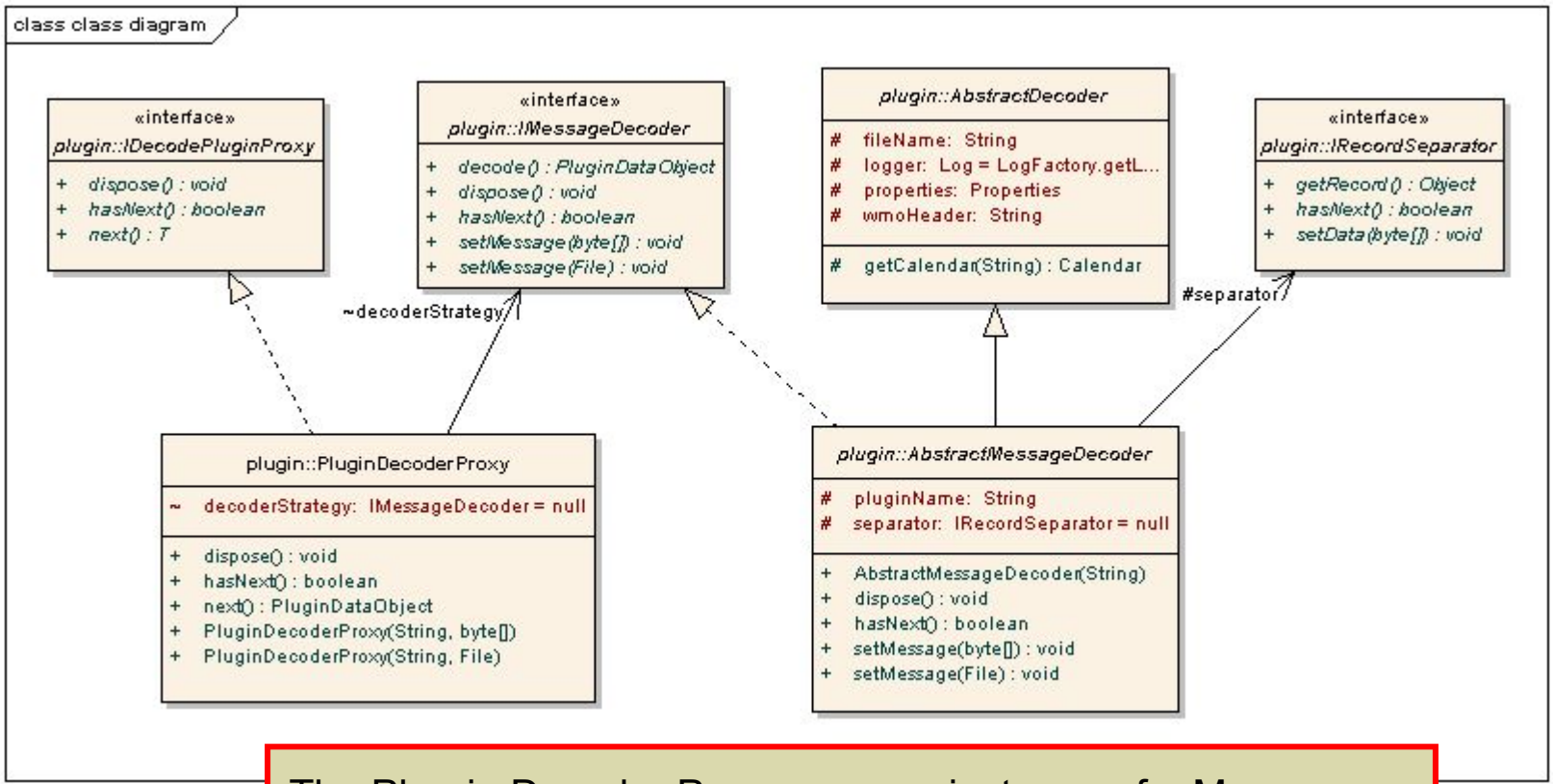
Raytheon

- Ingest Service (IngestSrv) uses the Plug-in Decoder Proxy to process various data types generically
  - IngestSrv.process() obtains a PluginDecoderProxy object to process the file
  - IngestSrv.process() calls the proxy's hasNext() method to determine if there is available data
  - IngestSrv.process() calls the proxy's next() method to obtain a single data record.

```
@Override
protected Object process() throws EdexException {
    this.message.setProperty("pluginName",pluginName);
    IDecodePluginProxy<PluginDataObject> proxy = null;
    List<PluginDataObject> retVal = new ArrayList<PluginDataObject>();
    byte[] msg = null;
    /*  Code used to obtain the data has been omitted */
    try {
        tableDef = Util.getTableDefinition(pluginName.toLowerCase());
        proxy = new PluginDecoderProxy(pluginName, msg);
        while (proxy.hasNext()) {
            PluginDataObject record = null;
            try {
                record = proxy.next();
            } catch (PluginException de) {
                /* Exception handler - omitted */
            }
            if (record != null) {
                record.setPluginName(pluginName);
                if(tableDef != null && tableDef.length > 0) {
                    record.constructDataURI(tableDef[0]);
                }
                if (record instanceof IPersistable) {
                    hdf5Dao.persistToHDF5(record);
                    record.setMessageData(null);
                }
                retVal.add(record);
                recCount++;
            }
        } // end while()
    } catch (Exception e) {
        /* Exception handler - omitted */
    } finally {
        if (proxy != null) {
            proxy.dispose();
        }
        /* Code for cleanup snd routine status logging omitted */
        if (retVal.size() == 0) {
            retVal = null;
        }
    }
    return retVal;
}
```

Note: Most logging and comments have been removed to save space.

Plug-In Architecture:
Decoder Proxy/Message Decoder Class Diagram

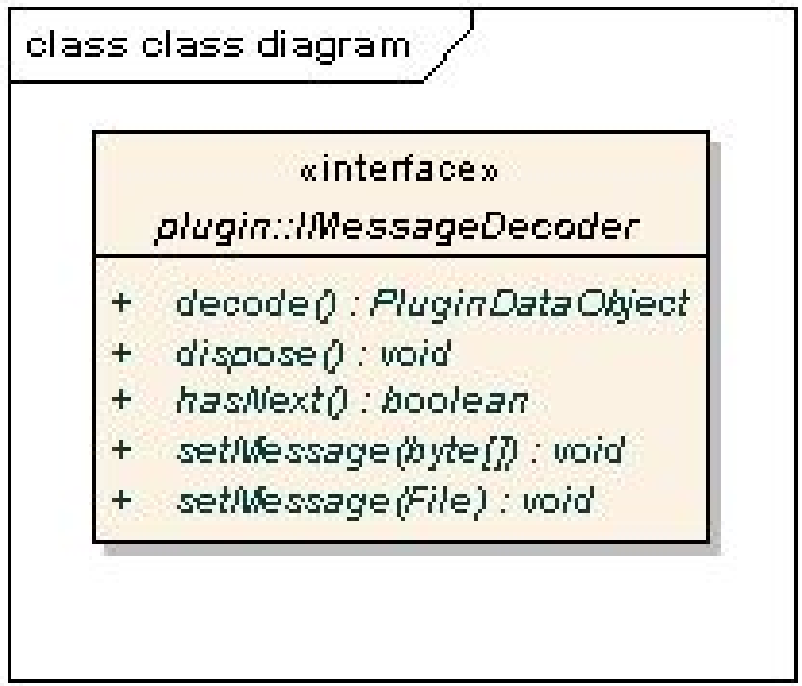


The Plug-in Decoder Proxy uses an instance of a Message Decoder to perform the message decoding. Each data-type Plug-In provides an implementation of IMessageDecoder that performs the actual decode operations.

Plug-In Architecture: IMessageDecoder Functionality

Raytheon

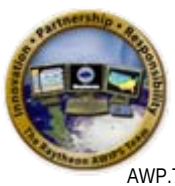
- IMessageDecoder interface specifies the methods used to decode a message and iterate the resulting records.
- The methods specified are:
  - setMessage() initializes the decoder with the message to decode
  - hasNext() determines if the message has a record to decode
  - decode() decodes the next record in the message
  - dispose() clean up any resources used by the decoder



Plug-In Architecture: IMessageDecoder Functionality (cont'd)

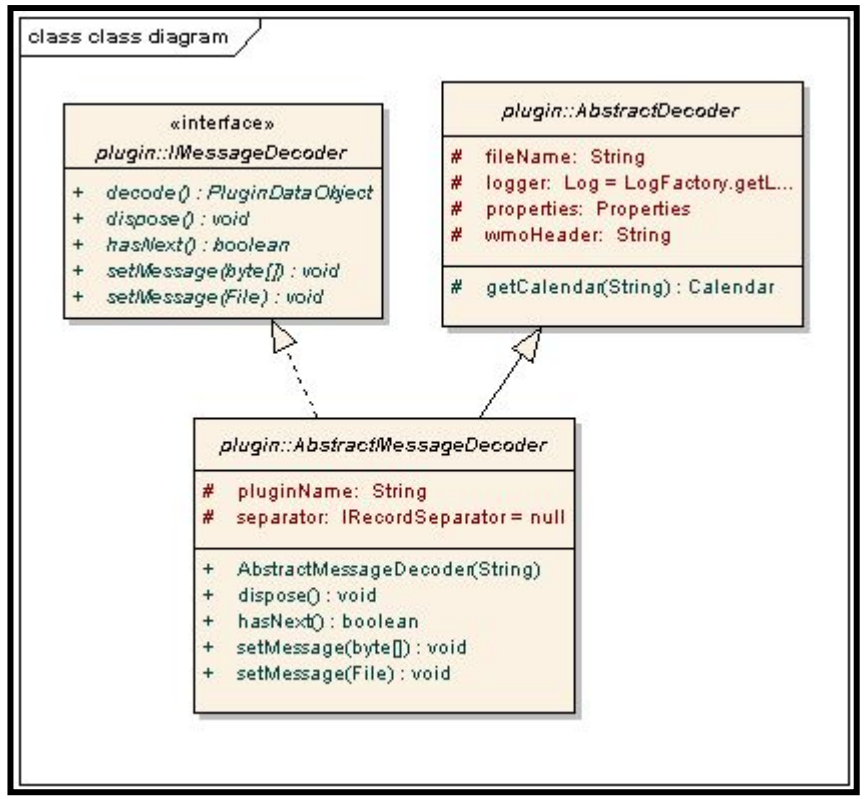
Raytheon

- Each Plug-In must provide a message decoder.
  - Plug-In's message decoder is obtained by the Plug-In Decoder Proxy using the Decoder Factory's get (…) method
    - The Decoder Factory is an instance of the DecoderFactory class.
  - Message decoder "lives" in the Plug-in Decoder Proxy. Its methods are called from that class.
- Note that the details of using the data-type specific decoder are hidden inside the Plug-In Decoder Proxy class and are not implemented within the data-type Plug-In itself.



Plug-In Architecture: Abstract Message Decoder

- Abstract base class implementing most of the IMessageDecoder functionality
  - Derives basic fields from the AbstractDecoder class.
  - Most data-type decoding functionality can be provided by extending the AbstractMessageDecoder base class
- Decoders extending the abstract base class must ensure that the message separator is set before any processing occurs
- Decoders must implement the decode() method – others are provided in the base class.
- In IngestSrv; decoders and decoder functionality is hidden behind the Plug-in Decoder Proxy



Plug-In Architecture: IMessageDecoder Utilization

Raytheon

- Calls to the IMessageDecoder methods are wrapped within PluginDecoderProxy methods.
  - Proxy's constructor calls the decoder's setMessage() method.
  - Proxy's hasNext() method calls the decoder's hasNext() method.
  - Proxy's next() method calls the decoder's decode() method.
  - Proxy's dispose() method calls the decoder's dispose() method.
- Things to note:
  - Decoder's hasNext() method acts as the controller for the loop processing multiple records
  - Decoder's dispose() method is called in a "finally" block.

```
@Override
protected Object process() throws EdexException {
    this.message.setProperty("pluginName",pluginName);
    IDecodePluginProxy<PluginDataObject> proxy = null;
    List<PluginDataObject> retVal = new ArrayList<PluginDataObject>();
    byte[] msg = null;
    /*  Code used to obtain the data has been omitted */
    try {
        tableDef = Util.getTableDefinition(pluginName.toLowerCase());
        proxy = new PluginDecoderProxy(pluginName, msg);
        while (proxy.hasNext()) {
            PluginDataObject record = null;
            try {
                record = proxy.next();
            } catch (PluginException de) {
                /* Exception handler - omitted */
            }
            if (record != null) {
                record.setPluginName(pluginName);
                if(tableDef != null && tableDef.length > 0) {
                    record.constructDataURI(tableDef[0]);
                }
                if (record instanceof IPersistable) {
                    hdf5Dao.persistToHDF5(record);
                    record.setMessageData(null);
                }
                retVal.add(record);
                recCount++;
            }
        } // end while()
    } catch (Exception e) {
        /* Exception handler - omitted */
    } finally {
        if (proxy != null) {
            proxy.dispose();
        }
        /* Code for cleanup snd routine status logging omitted */
        if (retVal.size() == 0) {
            retVal = null;
    }
    return retVal;
}
```

Note: Most logging and comments have been removed to save space.

Plug-In Architecture: IMessageDecoder Example

Raytheon

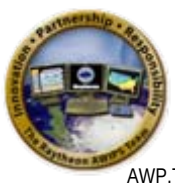
```
package gov.noaa.nws.edex.plugin.message;

import gov.noaa.nws.edex.util.message.MessageParser;

import com.raytheon.edex.db.objects.PluginDataObject;
import com.raytheon.edex.exception.DecoderException;
import com.raytheon.edex.plugin.AbstractMessageDecoder;
import com.raytheon.edex.plugin.RecordSeparatorImpl;

public class MessageDecoder extends AbstractMessageDecoder{
    @SuppressWarnings("unchecked")
    public MessageDecoder() throws DecoderException {
        super("MESSAGE");
        this.separator = new RecordSeparatorImpl();
    }
    @Override
    public PluginDataObject decode() throws DecoderException {
        Object object = this.separator.getRecord();
        MessageParser parser = new MessageParser((byte[])object);
        MessageRecord record = new MessageRecord();
        record.setHeader(parser.getHeader());
        record.setMessage(parser.getBody());
        return record;
    }
    @Override
    public void dispose() {
        super.dispose();
    }
}
```

Note: The setMessage() and hasNext() methods are provided by the base class.

This example shows the IMessageDecoder implementation in Message plug-in. The basic class code was generated by Eclipse using the ADE Plug-In Tool.

Code Examples: The following classes in the AWIPS EDEX baseline are implementations of IMessageDecoder:

- GribDecoder
- PirepDecoder
- ObsDecoder
- RadarDecoder
- SatelliteDecoder



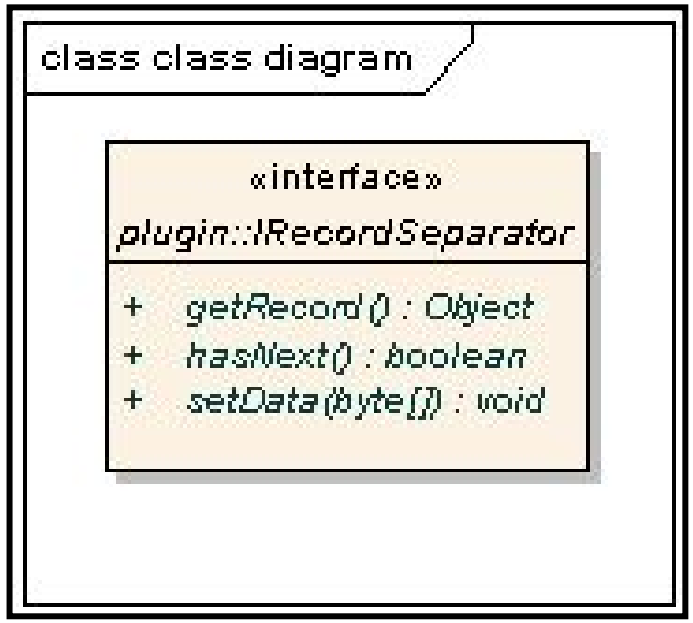
Plug-In Architecture: IRecordSeparator Functionality

Raytheon

- Defines functionality needed to separate data into individual records
  - Record Separator for a data type is utilized by the Plug-In's Data Decoder class to separate a multiple record message into single records
  - Record Separator is normally created when the Data Decoder is created and populated when the Data Decoder's setMessage(…) method is invoked
- Each Plug-In for data that contains multiple records must implement IRecordSeparator
  - Implementation details depend on the format of the data file
- Two options for a Plug-In for data that contains a single record:
  - Can provide a Record Separator
  - Can use RecordSeparatorImpl

Plug-In Architecture: IRecordSeparator Design

- IRecordSeparator specifies an “iterator” like interface via three methods:
  - setData() – sets the entire message into the Record Separator
  - hasNext() – determines if the message has another record to process
  - getRecord() – returns the next record in the message



Plug-In Architecture: IRecordSeparator Utilization

Raytheon

- In our example, the Message Decoder class uses the RecordSeparatorImpl class
  - Separator is created by the class constructor.
  - Decode() method uses the separator's getRecord() method to get the (single) record in the message.

```
package gov.noaa.nws.edex.plugin.message;

import gov.noaa.nws.edex.util.message.MessageParser;

import com.raytheon.edex.db.objects.PluginDataObject;
import com.raytheon.edex.exception.DecoderException;
import com.raytheon.edex.plugin.AbstractMessageDecoder;
import com.raytheon.edex.plugin.RecordSeparatorImpl;

public class MessageDecoder extends AbstractMessageDecoder{
    @SuppressWarnings("unchecked")
    public MessageDecoder() throws DecoderException {
        super("MESSAGE");
        this.separator = new RecordSeparatorImpl();
    }
    @Override
    public PluginDataObject decode() throws DecoderException {
        Object object = th.separator.getRecord();
        MessageParser parser = new MessageParser((byte[])object);
        MessageRecord record = new MessageRecord();
        record.setHeader(parser.getHeader());
        record.setMessage(parser.getBody());
        return record;
    }
    @Override
    public void dispose() {
        super.dispose();
    }
}
```

Note: Most logging and comments have been removed to save space.

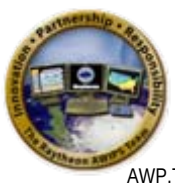
Plug-In Architecture: IRecordSeparator Utilization (cont'd)

Raytheon

- In the example, the Message Decoder class extends AbstractMessageDecoder base class
- The base class provides:
  - Separator field
  - setMessage(…) which calls the separator's setData(…) method
  - hasNext() which calls the separator's hasNext() method

```
package com.raytheon.edex.plugin;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import com.raytheon.edex.exception.DecoderException;
import com.raytheon.edex.util.PropertiesFactory;

public abstract class AbstractMessageDecoder extends AbstractDecoder
implements IMessageDecoder {
    protected IRecordSeparator separator = null;
    protected String pluginName;

    public AbstractMessageDecoder(String productType)
    throws DecoderException {
    pluginName = productType;
    properties = PropertiesFactory.getInstance().getPluginProperties(
                    productType);
    if (properties == null)
       throw new DecoderException("Unable to get plugin properties for: "
                        + productType);
    }

    @Override
    public void setMessage(byte[] messageData) {
        separator.setData(messageData);
    }

    @Override
    public void setMessage(File messageFile) {
        /* method details omitted to save space */
    }

    @Override
    public boolean hasNext() {
        return separator.hasNext();
    }

    public void dispose() {
    }
}
```

Note: Most logging and comments have been removed to save space.



Plug-In Architecture: RecordSeparatorImpl

Raytheon

- A Java class that provides a Record Separator for a data-type which consists of files containing single records
  - Package: com.raytheon.edex.plugin
- All IRecordSeparator methods are implemented
  - Calling setData() initializes the class
    - Prior to calling setData(), hasNext() returns “false”
  - Calling getRecord() returns the entire file
    - Prior to calling getRecord(), hasNext() returns “true”
    - After calling getRecord(), hasNext() returns “false”

Plug-In Architecture: Data Persistence Functionality

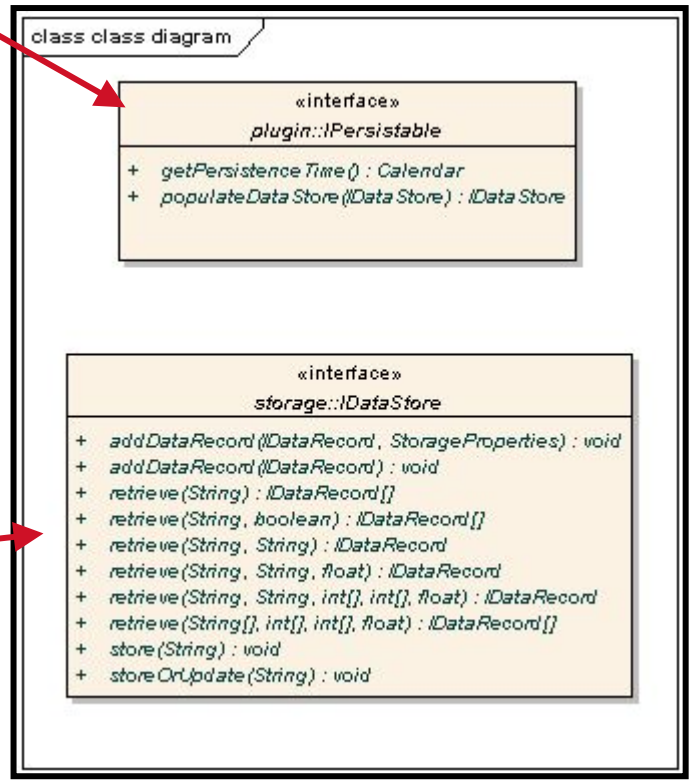
Raytheon

- Defines the functionality needed to persist the data from a decoded data message into the EDEX data store
  - EDEX uses HDF5 for data persistence
- A data-type Plug-In may elect to store data using HDF5
  - For Plug-Ins using HDF5 persistence, the Data Record class implements the HDF5 Persistence interface, IPersistable (part of the com.raytheon.edex.plugin package)
  - IngestSrv utilizes the methods provided by IPersistable to assist in writing the data to the HDF5 repository.

Plug-In Architecture: Data Persistence Functionality: IPersistable

Raytheon

- IPersistable specifies the functionality needed to persist data to the HDF5 repository
  - getPersistenceTime() returns a Calendar object representing the insert time of the data
  - populateDataStore(…) writes the IPersiable's data to a previously created IDataStore object representing the HDF5 repository
- Each HDF5 repository is represented by class that implements the IDataStore interface
  - IDataStore specifies methods used to access data in a data store



Plug-In Architecture: Data Persistence Functionality: IngestSrv

Raytheon

- IngestSrv checks the type of the data record to determine if the data is persisted to HDF5
  - If the data record implements the IPersistable interface, the contents are written to the HDF5 repository using the HDF5 DAO

Note: *The HDF5 DAO's use of IPersistable is discussed on the next slide.*

```
@Override
protected Object process() throws EdexException {
    this.message.setProperty("pluginName",pluginName);
    IDecodePluginProxy<PluginDataObject> proxy = null;
    List<PluginDataObject> retVal = new ArrayList<PluginDataObject>();
    byte[] msg = null;
    /*  Code used to obtain the data has been omitted */
    try {
        tableDef = Util.getTableDefinition(pluginName.toLowerCase());
        proxy = new PluginDecoderProxy(pluginName, msg);
        while (proxy.hasNext()) {
            PluginDataObject record = null;
            try {
                record = proxy.next();
            } catch (PluginException de) {
                /* Exception handler - omitted */
            }
            if (record != null) {
                record.setPluginName(pluginName);
                if(tableDef != null && tableDef.length > 0) {
                    record.constructDataURI(tableDef[0]);
                }
                if (record instanceof IPersistable) {
                    hdf5Dao.persistToHDF5(record);
                    record.setMessageData(null);
                }
                retVal.add(record);
                recCount++;
            }
        } // end while()
    } catch (Exception e) {
        /* Exception handler - omitted */
    } finally {
        if (proxy != null) {
            proxy.dispose();
        }
        /* Code for cleanup snd routine status logging omitted */
        if (retVal.size() == 0) {
            retVal = null;
    }
    return retVal;
}
```

Note: Most logging and comments have been removed to save space.

Plug-In Architecture: Data Persistence Functionality: HDF5Dao

Raytheon

- The HDF5 DAO's persistToHDF5(…) method writes the data to the HDF5 repository
  - The data record's getInsertTime is called to get the data's insert time
  - The data's insert time and Plug-In name are used to get access to the HDF5 repository for the data
  - The data record's populateDataStore(…) method is called to write the record's data to the HDF5 repository
  - The data record's insert time is saved back to the data record for future use

```
package com.raytheon.edex.db.dao;

// import statements omitted to save space

public class HDF5Dao {
   protected Log logger = LogFactory.getLog(getClass());
   protected String hdf5Dir = PropertiesFactory.getInstance()
         .getEnvProperties().getEnvValue("HDF5DIR");
   public void persistToHDF5(PluginDataObject record)
   throws DataAccessLayerException {
      if (record instanceof IPersistable) {
         IPersistable pRecord = (IPersistable) record;
         Calendar cal = record.getInsertTime();
         if (cal == null) {
            cal = pRecord.getPersistenceTime();
         }
         try {
            String hour = String.valueOf(cal.get(Calendar.HOUR_OF_DAY));
            String day = String.valueOf(cal.get(Calendar.DAY_OF_MONTH));
            String directory = hdf5Dir + "/" + day + "/" + hour;
            File file = new File(directory);
            if (!file.exists()) {
               file.mkdirs();
            }
            File dataStoreFile = new File(directory + "/"
                  + record.getPluginName().toLowerCase() + ".h5");
            HDF5DataStore dataStore = (HDF5DataStore) DataStoreFactory
                  .getDataStore(dataStoreFile);
            pRecord.populateDataStore(dataStore);
            dataStore.store((String) record.getIdentifier());
            record.setInsertTime(cal);
         catch (Exception e) {
            throw new DataAccessLayerException("Unable to persist "
                  + record.getPluginName().toUpperCase()
                  + " record to HDF5", e);
         }
      }
   }
   public IDataRecord retrieveFromHDF5(PluginDataObject obj) {
      IDataRecord record = null;
      /* method details omitted to save space */
      return record;
   }
   public IDataRecord[] retrieveGroupFromHDF5(PluginDataObject obj) {
      IDataRecord[] records = null;
      /* method details omitted to save space */
      return records;
   }
}
```

Note: Most logging and comments have been removed to save space.

Plug-In Architecture: IPersistable Example – SatelliteRecord

Raytheon

```
package com.raytheon.edex.plugin.satellite;

// import statements omitted

public class SatelliteRecord extends PluginDataObject
implements IPersistable {
    // class data fields omitted

    public SatelliteRecord() {
    }

    public SatelliteRecord(String uri, TableDefinition tableDef) {
        super(uri, tableDef);
    }

    @Override
    public IDataStore populateDataStore(IDataStore dataStore)
    throws Exception {
        AbstractStorageRecord storageRecord = null;
    long nx = ((SatMapCoverage) this.spatialInfo).getNx();
        long ny = ((SatMapCoverage) this.spatialInfo).getNy();
        long[] sizes = new long[] { nx, ny };
        storageRecord = new ByteDataRecord("Data",
                (byte[]) this.messageData,
                2, sizes);
        StorageProperties props = new StorageProperties();
        props.setChunked(nx > HDF5DataStore.CHUNK_SIZE
                && ny > HDF5DataStore.CHUNK_SIZE);
        props.setDownscaled(true);
        storageRecord.setProperties(props);
        dataStore.addDataRecord(storageRecord);
        return dataStore;
    }

    @Override
    public Calendar getPersistenceTime() {
        return Calendar.getInstance(TimeZone.getTimeZone("GMT"));
    }

    // field accessors omitted
}
```

Note: Class fields, most logging, and comments have been removed to save space.

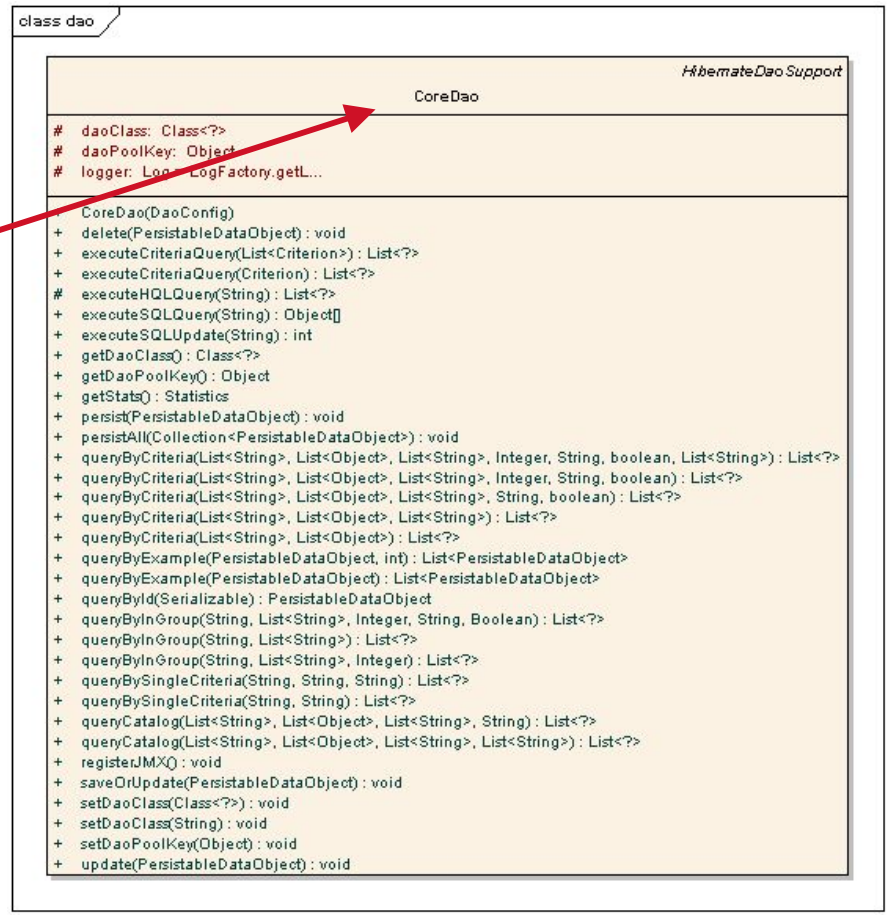
- This example shows how IPersistable's methods are implemented in the Satellite plug-in's data record, SatelliteRecord
- Code Examples: The following classes in the AWIPS EDEX baseline are implementations of IPersistable:
  - BinLightningRecord
  - GribRecord
  - RadarRecord

Note: *Our example Plug-In does not use HDF5 persistence*

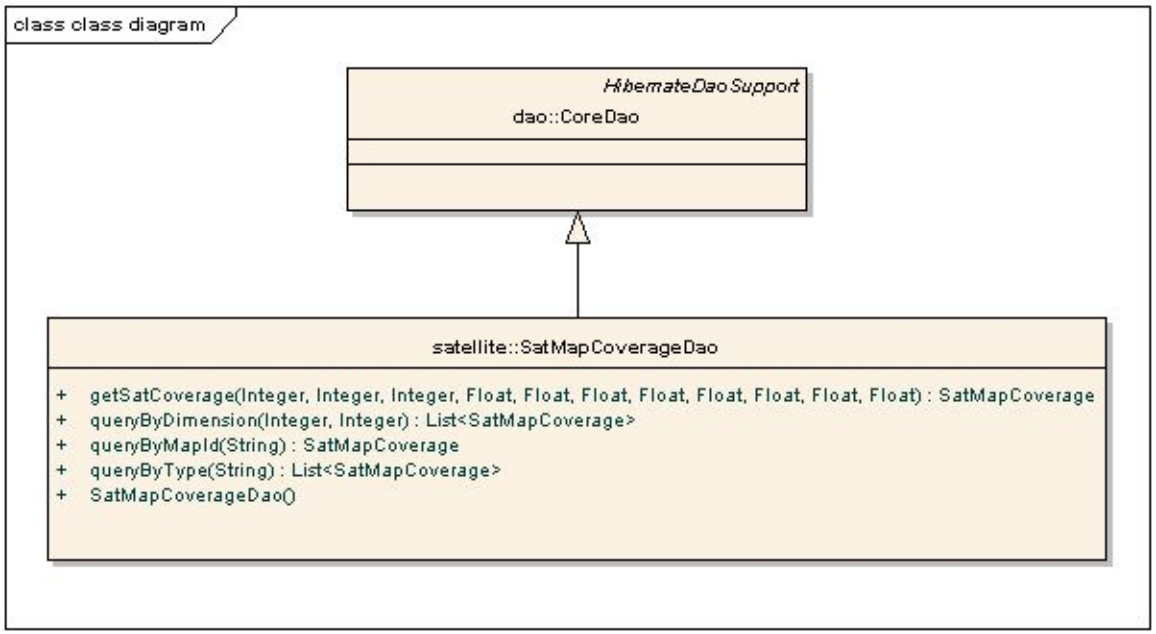
Plug-In Architecture: Data Access Objects

Raytheon

- A data-type Plug-In persists meta-data to the AWIPS II Metadata database
  - In most cases, database persistence can be handled via the Core DAO (CoreDao)
- If a Plug-In needs special database handling, e.g., as part of decoding, it can provide one or more DAOs
  - The specialty DAO should extend CoreDAO and may override one or more CoreDao methods
- Data Access Objects provided in a data-type Plug-In are not normally accessed outside the Plug-In

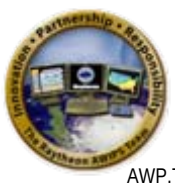


Plug-In Architecture: Data Access Objects - SatMapCoverageDao



- The satellite data-type Plug-In includes the SatMapCoverageDao
- It provides query methods used by the satellite decoder to obtain map coverage information from the database
  - This DAO provides wrappers for CoreDao methods as well as convenience methods not provided by CoreDAO
- This DAO specifically provides access to the Satellite Map Coverage data in the EDEX database

Data Record Objects



Plug-In Architecture: Data Record Functionality

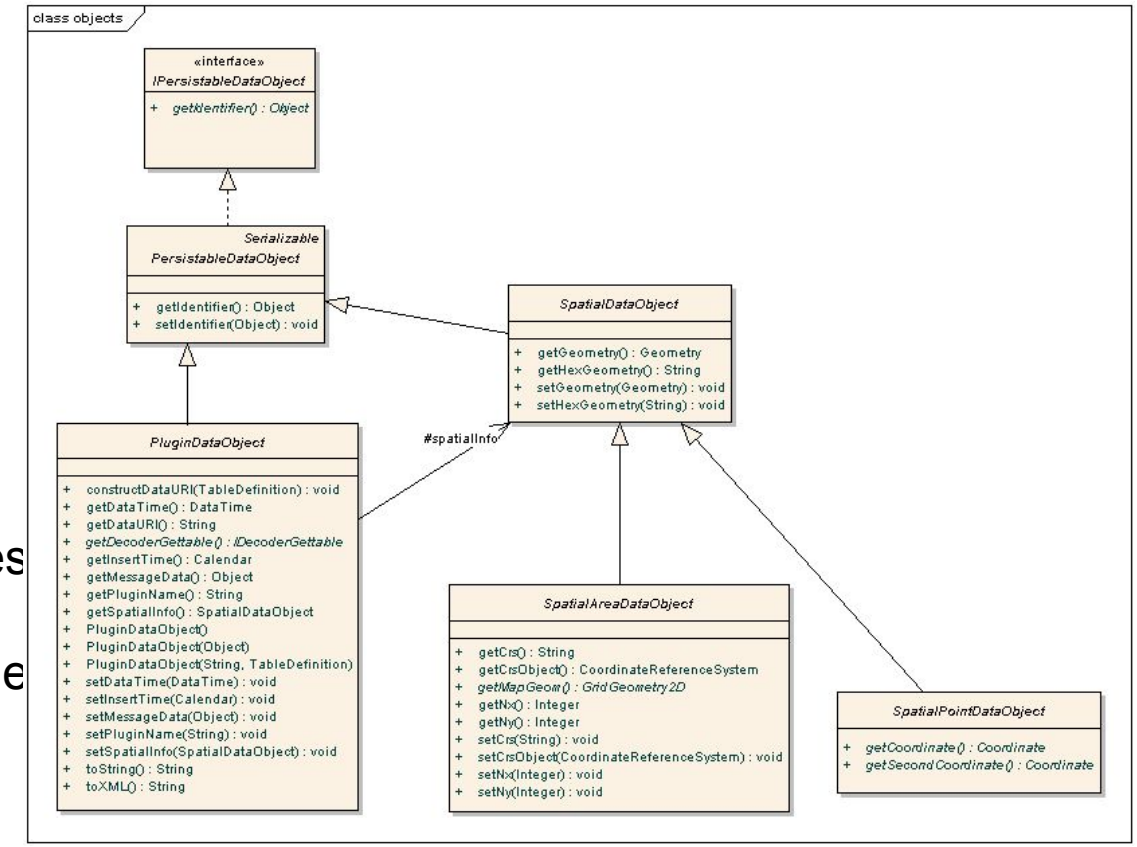
Raytheon

- Data Records:
  - Provide the internal representation of the metadata and data produced by the data decoder
  - Implement the PluginDataObject abstract Java class, which provides the basic fields required for EDEX data persistence including a field for the message data
- Each AbstractDataRecord implementation:
  - Provides a set of fields for the data-type's metadata
    - Additional fields may be provided decoded data values that are not part of the metadata
  - Must provide an implementation of the getDecoderGettable() method
  - May implement the IPersistable interface (discussed previously)

Plug-In Architecture: Data Record Design

Raytheon

- This diagram shows the structure of the PluginDataObject class and associated classes
  - PersistableDataObject provides the minimal functionality required for database persistence
  - SpatialDataObject provides the basic functionality for including apatial data in the data record – subclasses are provided for point and area data.



- The data Record implementation provides fields, with accessors, for the data-type specific values

Plug-In Architecture: Data Record Utilization

Raytheon

This example shows how the MetarDecoder creates and starts to populate a MetarRecord object.

For details on how the decoding is performed, refer to the MetarDecoder source code in the EDEX baseline.

```
public MetarRecord decode() throws DecoderException {
   byte[] messageData = null;
   if (separator.hasNext()) {
      messageData = (byte[]) separator.getRecord();
   } else {
      throw new DecoderException("Out of data");
   }
   theMessage = new String(messageData);
   MetarRecord record = new MetarRecord(theMessage);
   theMessage = cleanMessage(theMessage);

   // Additional code ito decode the METAR
   // and populate the record - omitted
}
```

Plug-In Architecture: Data Record Example

Raytheon

This example shows the AbstractDataRecord implementation in MessagePlug-in.

Note that this class extends AbstractTextDataRecord since a FooBar record is an ascii text-based data record.

The basic class code was generated by Eclipse using the ADE Plug-In Tool.

Code Examples: The following classes in the AWIPS EDEX baseline are implementations of AbstractDataRecord:

- GribRecord
- RadarRecord
- SatelliteRecord
- MesowestRecord
- MetarRecord

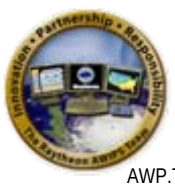
```
package com.raytheon.edex.plugin.message;

import
com.raytheon.edex.plugin.data.AbstractTextDataRecord;

public class MessageRecord extends AbstractTextDataRecord {

    private static final long serialVersionUID = 1L;

    private String header;

    private String body;

    public String getBody() {
        return body;
    }

    public void setBody(String body) {
        this.body = body;
    }

    public String getHeader() {
        return header;
    }

    public void setHeader(String header) {
        this.header = header;
    }

}
```

Note: Most logging and comments have been removed to save space.

BREAK


Innovation • Partnership • Responsibility
The Raytheon Mission Team

Data Plug-In Configuration



Plug-In Architecture: Required Configuration Files

Raytheon

- Performance and availability of an EDEX Plug-In determined by a set of five configuration files:
  - plugin.xml. Basic file mapping the Plug-In implementation classes to the Plug-In for use by the Plug-In Factory
  - <plugin-name>-ingest.xml. Defines the Mule ingest end points for the Plug-In
  - binding.xml. Optional file defining JiBX mappings for data objects in the Plug-In.
  - <data-name>.db.xml. File defining PostgreSQL metadata table structure as required by the Plug-In.
  - <data-name>.hbm.xml – File mapping the PostgreSQL metadata table to the Data Record class for Hibernate.

Note: *The plug-in may include multiple .db.xml and .hbm.xml*.

Plug-In Configuration: plugin.xml

Raytheon

- pluxin.xml provides the mapping of the Plug-In's components to attribute names used by the Plug-In Factory for class creation
- XML provides:
  - The plug-in name
  - The required interface implementations
  - Additional attributes used by the plug-in

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<properties>
   <Name>MESSAGE</Name>
   <Plugin>true</Plugin>
   <Decoder>gov.noaa.nws.edex.message.MessageDecoder</Decoder>

   <!-- plug-in specific attribute values -->

</properties>
```

Note: *See the other EDEX Plug-Ins for additional examples.*

Plug-In Configuration: attributes.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<properties>
<Name>ATTRIBUTE_NAMES</Name>

<!-- Values used by the plug-in -->

</properties>
```

- attributes.xml defines the configuration attribute names for use by the plug-in
  - Name tag contains “ATTRIBUTE_NAMES”
  - Other tags may be added as needed

Note: *This is an optional configuration file.*

Plug-In Configuration: binding.xml

Raytheon

- Binding.xml defines the JiBX bindings for the Plug-In

```
<?xml version="1.0" encoding="UTF-8"?>

<binding>
   <!-- Mappings for a message record -->
   <mapping name="MESSAGE" class="gov.noaa.nws.edex.message.MessageRecord">
      <!-- from PersistableDataObject -->
      <value name="dataURI" field="identifier" usage="optional"
         deserializer="com.raytheon.edex.util.ObjectToStringConv.deserializer" />
      <!-- from PluginDataObject -->
      <value name="pluginName" field="pluginName" usage="optional" />
      <structure field="dataTime"
         marshaller="com.raytheon.edex.plugin.marshaller.DataTimeMarshaller"
         unmarshaller="com.raytheon.edex.plugin.marshaller.DataTimeMarshaller" />
      <value name="insertTime" field="insertTime" usage="optional"
         serializer="com.raytheon.edex.util.CalendarConv.serializer"
         deserializer="com.raytheon.edex.util.CalendarConv.deserializer" />
      <structure field="spatialInfo" />
      <!-- from MessageRecord -->
      <value name="header" field="header" usage="optional" />
      <value name="message" field="message" usage="optional" />
   </mapping>

</binding>
```

This example shows the bindings file for the MessagePlug-in.

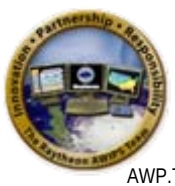
Note that most of the fields are inherited from two base classes: AbstactData Record and AbstractTextDataRecord.

In each “value” tag:

Name attribute defines the field name in the XML.

Field attribute defines the class field to capture and must match the field in the class.

Setting usage to “optional” allows the binding to work on partially populated Classes.



Plug-In Configuration: <data-name>.db.xml

Raytheon

- <data-name>.db.xml defines the metadata table structure for the PostgreSQL database
- Items defined:
  - The table name
  - The data class represented
  - Data retention time
  - Definition for table columns

```
<?xml version="1.0" encoding="UTF-8"?>
<tableDefinition>
    <tableName>obs</tableName>
    <hibClass>com.raytheon.edex.plugin.obs.metar.MetarRecord</hibClass>
    <retentionHours>24</retentionHours>
    <order>1</order>
    <defaultClass>true</defaultClass>
    <partitioned>true</partitioned>
    <linkedExternally>false</linkedExternally>

    <columnDefinition>
        <name>datauri</name>
        <columnType>varchar</columnType>
        <constraintType>PRIMARY KEY</constraintType>
        <precision>256</precision>
        <index>none</index>
    </columnDefinition>

    <columnDefinition>
        <name>message</name>
        <columnType>varchar</columnType>
        <constraintType>none</constraintType>
        <precision>2048</precision>
        <index>none</index>
    </columnDefinition>

    <columnDefinition>
        <name>dataTime</name>
        <columnType>varchar</columnType>
        <constraintType>none</constraintType>
        <precision>256</precision>
        <scale>0</scale>
        <dataURI>true</dataURI>
        <index>none</index>
    </columnDefinition>
    <!-- additional column definitions -->
</tableDefinition>
```

This example shows a partial definition of the main database table used for the METAR Plug-In's metadata. The child tables are defined in separate table definitions.

Plug-In Configuration: <data-name>.db.xml

```
<columnDefinition>
    <name>datauri</name>
    <columnType>varchar</columnType>
    <constraintType>PRIMARY KEY</constraintType>
    <precision>256</precision>
    <index>none</index>
</columnDefinition>
```

- Contents of column definition island in the table definition file:
  - Name: The name of the column
  - columnType: The data type for the column, e.g., integer, varchar
  - constraintType: The constraint on the table, e.g., “PRIMARY KEY” or “UNIQUE,” “none”
  - Precision: The size of the field
  - Index: Specifies whether an index should be created on this column

Note: *Details on creating this file are included in the ADE documentation.*

Plug-In Configuration: <data-name>.hbm.xml

Raytheon

- <data-name>.hbm.xml defines the mapping of Data Record class to PostgreSQL database meta data table for Hibernate

Note: *Details on creating the Hibernate file are included in the ADE documentation.*

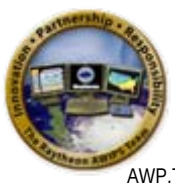
```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping default-lazy="false" schema="awips">
   <class name="com.raytheon.edex.plugin.obs.metar.MetarRecord" table="obs">
      <id name="identifier" column="datauri" type="java.lang.String"/>
      <property name="dataTime" column="dataTime"
          type="com.raytheon.edex.db.objects.hibernate.DataTimeType"/>

      <!-- these are typical field definitions -->
      <property name="reportType" column="reportType" type="java.lang.String"/>
      <property name="stationID" column="stationID" type="java.lang.String"/>
      <property name="stationLat" column="stationLat" type="java.lang.Double"/>
      <property name="stationLon" column="stationLon" type="java.lang.Double"/>

      <property name="messageData" column="message" type="java.lang.String"/>
      <property name="timeObs" column="timeobs" type="java.util.Calendar"/>
      <property name="autoStationType" column="autostationtype"
         type="java.lang.String"/>

      <!-- definition of a link to a secondary table -->
      <many-to-one name="spatialInfo" insert="false" update="false"
          class="com.raytheon.edex.db.objects.spatial.ObStation" column="stationID"
          property-ref="icao" cascade="none"/>

      <!-- definition of a link to a secondary table -->
      <property name="skyKey" column="skycoverage" type="java.lang.String"/>
      <set name="skyCoverage" inverse="true" cascade="all">
         <key column="parentid"/>
         <one-to-many class="com.raytheon.edex.plugin.obs.metar.util.SkyCover"/>
      </set>

      <!-- definitions of additional fields have been omitted -->
   </class>

   <!-- definitions of contained classes - details omitted -->
   <class name="com.raytheon.edex.plugin.obs.metar.util.SkyCover"
          table="skycover">
   </class>
   <class name="com.raytheon.edex.plugin.obs.metar.util.WeatherCondition"
          table="weather">
   </class>
</hibernate-mapping>
```

This example is a partial listing of the Hibernate file for the METAR Record class used by obs Plug-in.

BREAK


Innovation • Partnership • Responsibility
The Raytheon Warrior Team

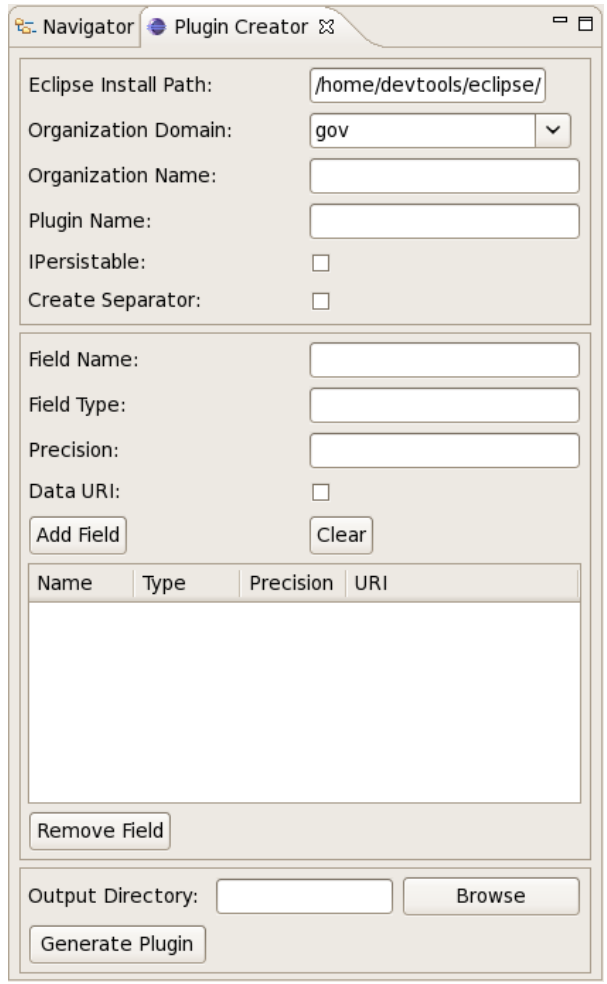
ADE Plug-In Creation Tool



AWIPS Plug-In Creation Tool: New With ADE 1.0

AWIPS Plug-In Creation Tool

- Implemented as an Eclipse Plug-In
- Once installed, may be used like any other element of the Eclipse IDE
- Automates most of the work required to generate the initial files for a data-type Plug-In
  - Automatically creates the required directory structure for the Plug-In
  - Generates class stubs for the required Java classes
  - Generates initial configuration and build files



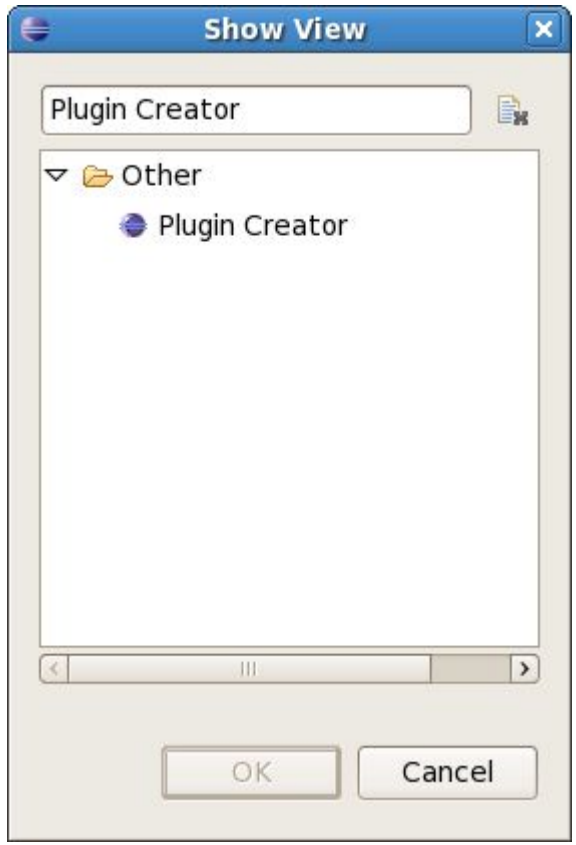
Installation of ADE Plug-In Creation Tool

- Plug-In Creation Tool ships with the ADE EDEX baseline, located in opt/tools/plugins
  - The file: com.raytheon.edex.pluginCreator_1.0.0.jar
- To install the Plug-In Creation Tool:
  - Copy the jar file into the plugins directory in your Eclipse installation
  - Restart Eclipse
- When Eclipse restarts, the Plug-in Creation Tool will be available

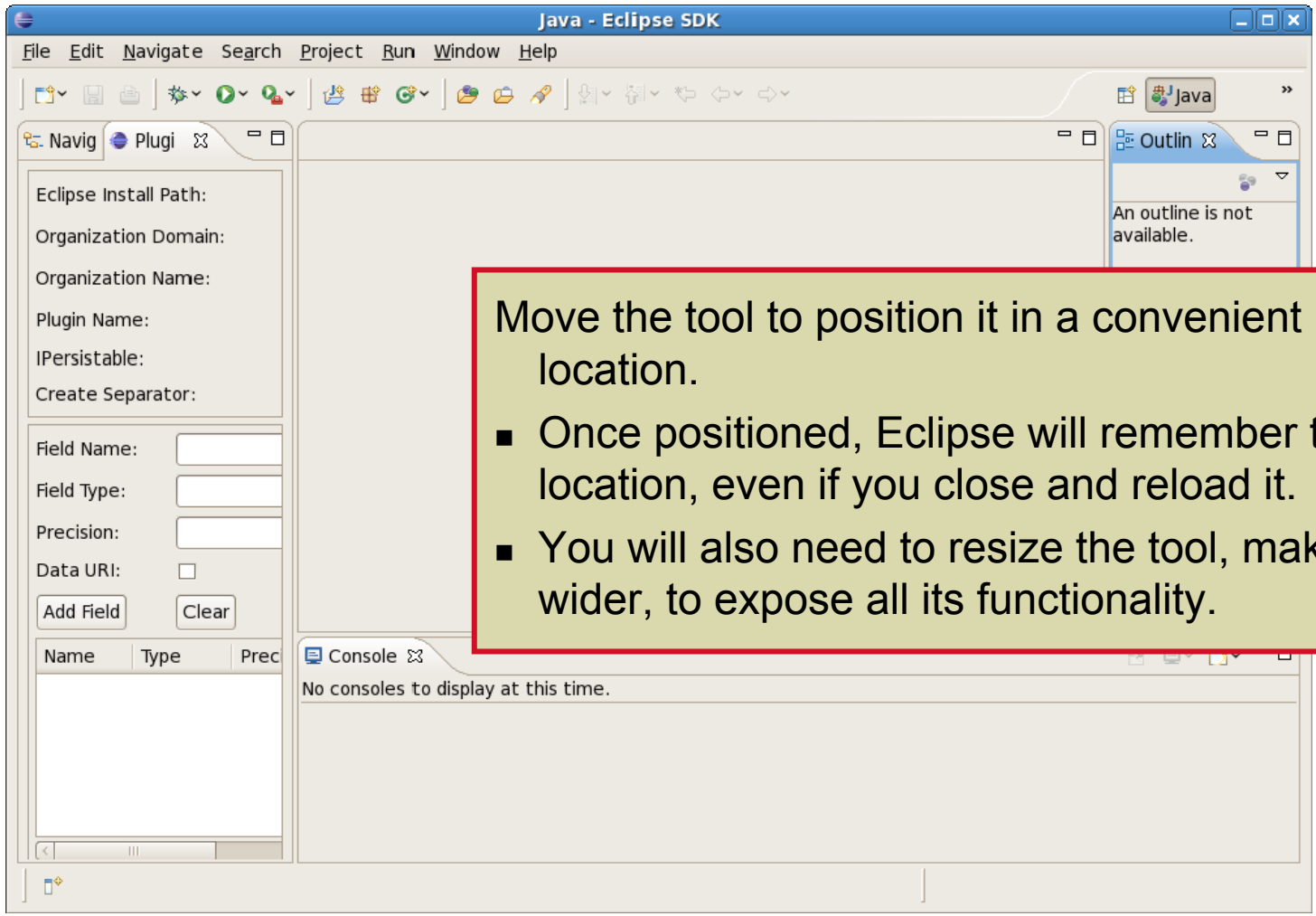
Accessing ADE Plug-In Creation Tool

To access the Plug-In Creation Tool:

- Select *Show View→Other…* from the *Window* menu to display the *Show View* dialog
- On the *Show View* dialog, type *Plugin Creator* in the text box (this will locate the Plugin Creator icon)
- Select the Plugin Creator icon and click OK. This adds the Plugin Creator to your Eclipse



Accessing ADE Plug-In Creation Tool (cont'd)



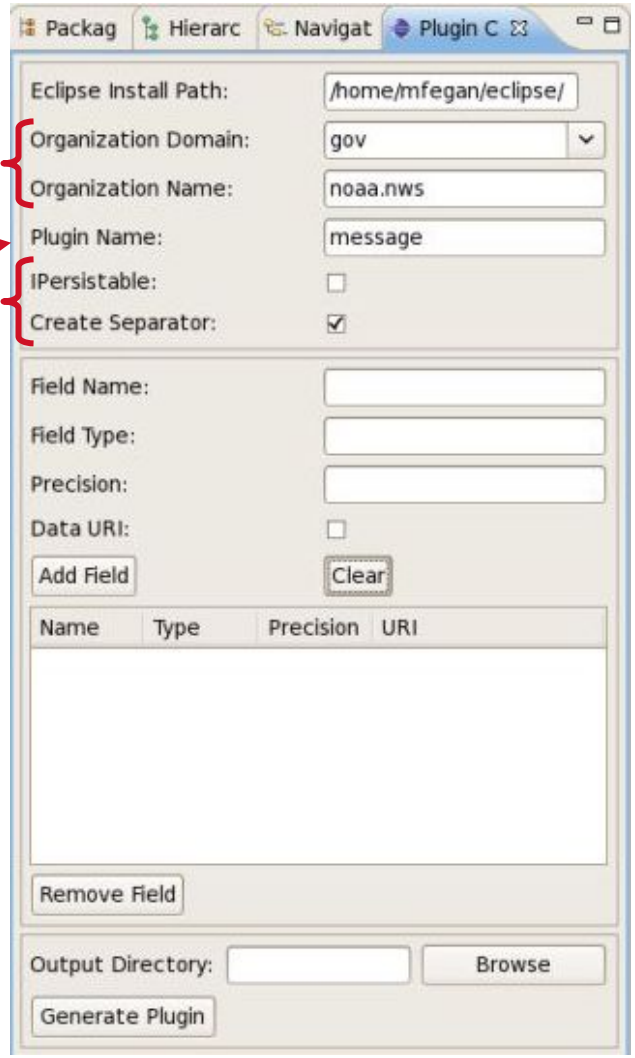
Move the tool to position it in a convenient location.

- Once positioned, Eclipse will remember the location, even if you close and reload it.
- You will also need to resize the tool, making it wider, to expose all its functionality.

Creating a Plug-In With the Tool

- Enter the eclipse install path
- Enter basic Plug-In information
  - Organization name and domain
    - Domain may include dots
  - Plug-In name
- Enter auxiliary information
  - Check “IPersistable” if the Plug-In will persist data to HDF5
  - Check “Create Separator” if the Plug-In should implement a data separator.

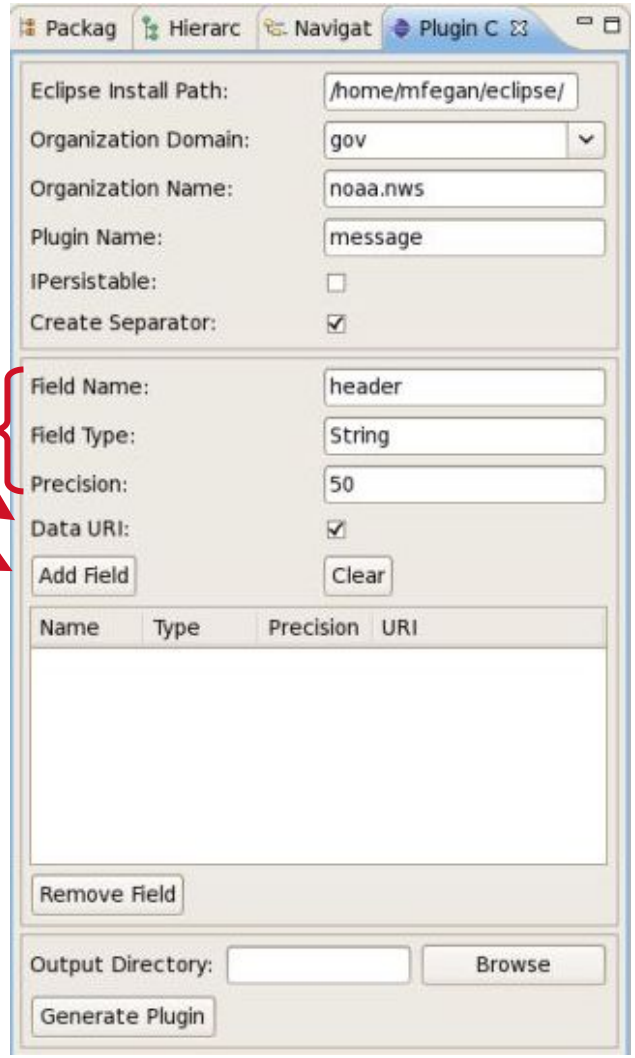
Note: *The message Plug-in, which is a text based product, does not persist to HDF5*



Creating a Plug-In With the Tool (cont'd)

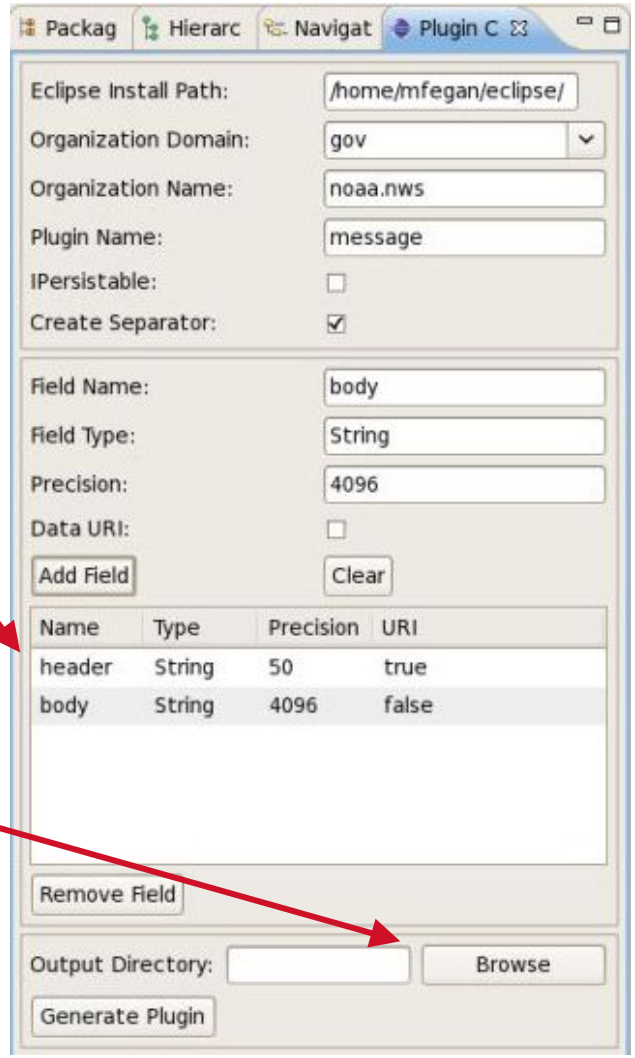
- Specify the fields for the data representation:
  - Enter the field name and type information
  - Click to include in data URI
  - Add the field to the plug-in definition
- Caution: Use care when deciding which fields to include in the data URI.
  - Include enough fields to uniquely identify the data.

Note: *The data URI provides a unique identifier of the data in the PostgreSQL database, and acts as the HDF5 pointer for binary data.*



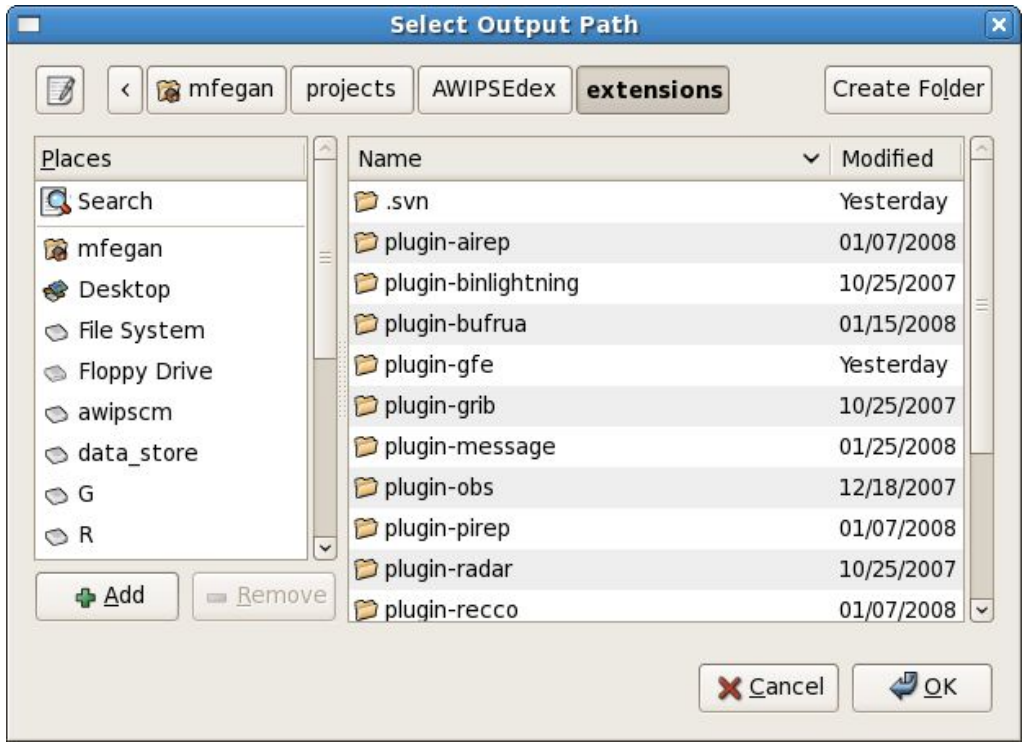
Creating a Plug-In With the Tool (cont'd)

- For the message Plug-In, there are two data fields: header and body. Because the header uniquely identifies the message, no other fields are needed for the data URI.
- Once the fields for the data have been added, use the browse button to display the *Select Output Path* dialog and select the location to create the Plug-In.



Creating a Plug-In With the Tool (cont'd)

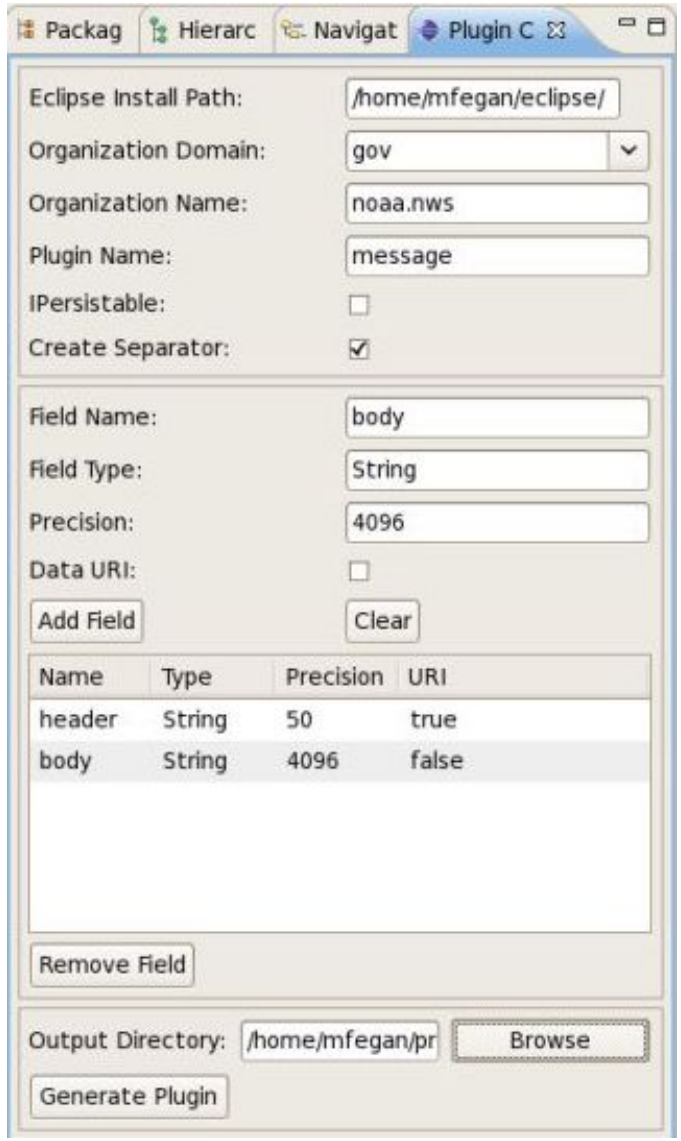
- The *Select Output Path* dialog provides a means to browse for the desired output directory. ***Note:*** *The plug-in is created in a subdirectory of the directory you select.*
- For the "message" Plug-In, we will create the code in the EDEX extensions directory. The folder created will be *plugin-message*.



Creating a Plug-In With the Tool (cont'd)

- Generate the Plug-in code:
  - Click the *Generate Plugin* button
- Once the Plug-In has been generated, close the Plug-In Creation Tool

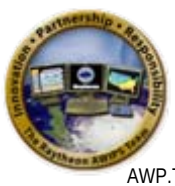
Note: *In order for Eclipse to pick up the new code, you will need to refresh the project display.*



Creating a Plug-In With the Tool (cont'd)

- Once the plug-in has been created using the tool, any code needed to provide functionality may be added. This includes code for any helper classes and for any μEngine tasks.
- ***One final reminder:*** The name of the Plug-In must be added to the deployment properties file
  - See AWIPS EDEX Build Files To Modify

Adding Plug-In to EDEX Ingest



Plug-In Architecture: Adding the Ingest End Point

Raytheon

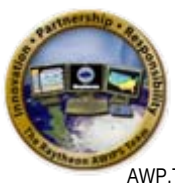
- In this section: Details for adding the data-type to the Ingest component
  - Consists of setting up Mule end points to ingest the data
- To use the Plug-In for data ingest, add a Mule descriptor to its ingest configuration file
  - <plugin-name>-ingest.xml is located in res/endpoints in the plug-in directory
- <plugin-name>-ingest.xml is generated by the plug-in creation tool
  - Normally, no changes are needed to the tool-generated file

Plug-In Architecture: Adding the Ingest End Point

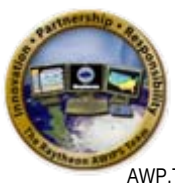
Raytheon

- This example shows the ingest definition for the message Plug-In as generated by the Plug-In tool
- Some changes made to the Ingest specification in TO 8
- More details on the modified specification available in Module 12; *TO8 ADE 1.0 Developer Updates*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mule-configuration PUBLIC "-//MuleSource //DTD mule-configuration XML V1.0//EN"
    "http://mule.mulesource.org/dtds/mule/mule-1.4.0-spring-configuration.dtd">

<mule-configuration version="1.0">
   <!-- Endpoint to stage MESSAGE Data -->
   <model name="edex" type="seda">
      <mule-descriptor name="Awips.Mule.Service.StagingSrv-message"
          singleton="true" implementation="com.raytheon.edex.services.StagingSrv"
          outboundEndpoint="jms://ar/message">
         <inbound-router>
            <endpoint name="messageIngestEndpoint"
               address="file://../../data/sbn/message/?transformers=FileToString">
               <properties>
                  <property name="autoDelete" value="false" />
                  <property name="moveToDirectory" value="../../processing" />
               </properties>
            </endpoint>
         </inbound-router>
         <threading-profile maxThreadsActive="1" maxThreadsIdle="1" />
      </mule-descriptor>

      <!-- Endpoint to archive MESSAGE Data -->
      <mule-descriptor name="Awips.Edex.Service.ArchiveSrv-message"
         singleton="false" implementation="com.raytheon.edex.services.ArchiveSrv"
         outboundEndpoint="jms://cp/message">
         <inbound-router>
            <endpoint name="AR-Message" address="jms://ar/message" />
         </inbound-router>
         <threading-profile maxThreadsActive="4" maxThreadsIdle="4" />
         <properties>
            <property name="pluginName" value="MESSAGE" />
            <property name="archiveDirectoryLocation" value="../../data/archive/message/" />
            <property name="jmxModeOn" value="true" />
         </properties>
      </mule-descriptor>

      <!-- Endpoint to ingest MESSAGE Data -->
      <mule-descriptor name="Awips.Edex.Service.IngestSrv-message"
         singleton="false" implementation="com.raytheon.edex.services.IngestSrv">
         <inbound-router>
            <endpoint name="CP-Message" address="jms://cp/message" />
         </inbound-router>
         <outbound-router>
            <router className="org.mule.routing.outbound.FilteringListMessageSplitter">
               <endpoint address="vm://indexVMQueue" />
            </router>
         </outbound-router>
         <threading-profile maxThreadsActive="4" maxThreadsIdle="4" />
         <properties>
            <property name="pluginName" value="MESSAGE" />
         </properties>
      </mule-descriptor>
   </model>
</mule-configuration>
```



Adding uEngine Data-Type Processing

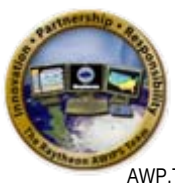


Plug-In Architecture: Adding a uEngine Task

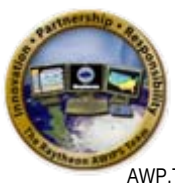
Raytheon

- A data-type Plug-In that requires special processing may include data-type specific uEngine tasks
  - uEngine tasks extend the abstract ScriptTask class
- For basic data retrieval, the uEngine provides many of the tasks needed to process data
  - For ASCII, point data tasks exist to convert the data to XML format for retrieval
  - For blob data such as GRIB data, the uEngine includes tasks for processing the data and converting it to visualization products
- Specialized tasks may be created for specific data types
  - Example: For the e-mail message discussed in the homework, it might be desirable to create a uEngine task that would send the email

Note: *Creating a uEngine task is beyond the scope of this tutorial. For more information, see Module 3, MicroEngine Scripting.*



Wrap-Up



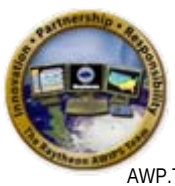
Summary

- What we just learned
- Future evolution of the Plug-In design
- Homework and where to get the solutions

Resources

- On the ADE 1.0 DVD
  - Current Plug-In code available for examination in the EDEX baseline
  - JavaDoc documentation for Plug-Ins available
  - AWIPS EDEX design documents – location TBD
  - AWIPS ADE 1.0 Training Materials – location TBD

Lab Work / Home Work



Hands-On Exercise: Data Type Plug-In

Instructions

1. Review the code for the Satellite Plug-In.
2. Test the Plug-In by ingesting data and retrieving products.
3. Complete the code for the Message Plug-in.
4. Write a Plug-In to handle “email-like” messages arriving on EDEX. Assume that each message has two main parts – the message header and the message body. The message header will have five semicolon-terminated lines: Date; sender; addressee; cc address; subject. The body will have zero or more lines terminating with an end-of-message token (three equal signs) on a separate line. Each file may contain multiple messages. (Sample messages appear on the next slide.)

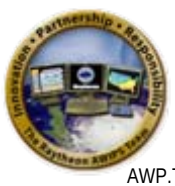
Hands-On Exercise: Data Type Plug-In (cont'd)

Sample Messages for the Final Exercise

1. A complete message. Note that each header element is on a separate line.
2. A message with a missing header element. The missing element is denoted by a line containing only a semicolon.

```
14 February 2007;
your.name@your.address;
send.to.name@send.to.address;
copy.to.name@copy.to.address;
subject line of the message;
the body of the message, may
be more than one line, it is
terminated by a triple equal
sign (===) on a separate line.
===
```

```
14 February 2007;
your.name@your.address;
send.to.name@send.to.address;
;
subject line of the message;
the body of the message, may
be more than one line. Note
that there is no copy-to
address for this message.
===
```



Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 5: Service Oriented Architecture (SOA) (rev.1)

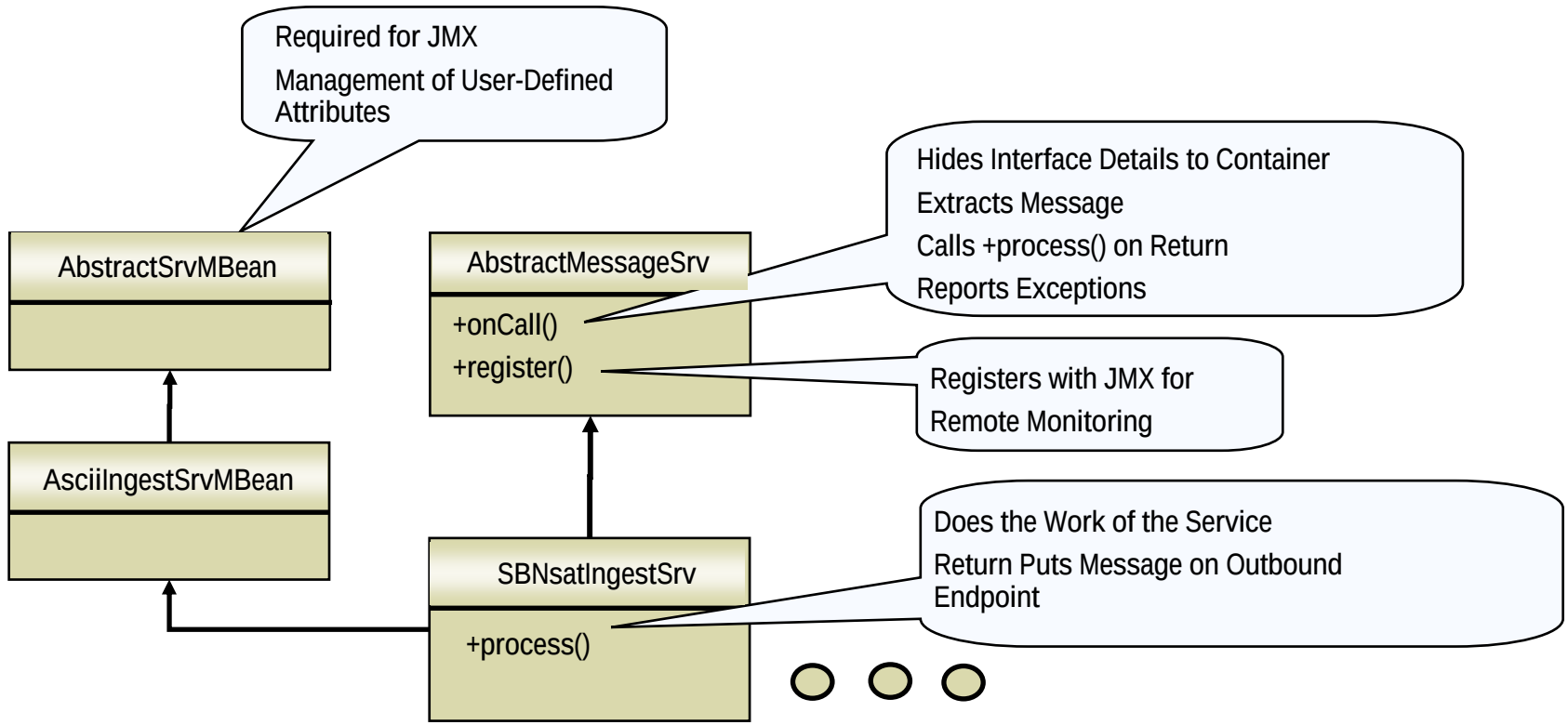
February 14, 2008

AWP.TRG.SWCTR/TO6.ADE/CAVE05.01

Objectives

- Understand the architectural pattern of a Service Oriented Architecture (SOA) service
- Understand how services are written
- Understand how services are integrated into the system
- Understand how to monitor and test an SOA service

SOA Service Design Pattern: Wrap the Interface to the ESB/Container



ESB Message Event Handling to Service

AbstractMessageSrv
+onCall() +register()

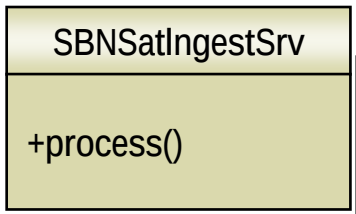
```
public Object onCall(UMOEventContext eventContext) throws Exception {
    logger.debug("Received a message");

    // set the event context
    this.eventContext = eventContext;

    // get the message from the context
    message = eventContext.getMessage();

    // get the original file name
    fileName = message.getStringProperty(
            FileConnector.PROPERTY_ORIGINAL_FILENAME, "");

    // append directory information to the file name
    fileName = message.getStringProperty(FileConnector.PROPERTY_DIRECTORY,
            "")
            + File.separator + fileName;

    Object retVal = null;
    try {
        retVal = process();
    } catch (EdexException e) {
        logger.error(this.getClass() + " failed to process message", e);
    }

    return retVal;
}
```

Reference to Mule ESB events

Extracts message out of Mule ESB event

Executes +process() Method of sub class

SOA's Service Process Method




```
protected Object process() throws EdexException {
    Object returnVal = null;
    this.message.setProperty("productType", "Imagery");

    //get the file as a byte array
    byte[] msg = null;
    try {
        msg = this.message.getPayloadAsBytes();
    } catch (Exception e) {
        throw new EdexException("Unable to receive ingested file as a byte array", e);
    }

    logger.info("FileIngestSrv received a message with: " + this.fileName
            + "len: " + msg.length);

    //process the message
    long begin = System.currentTimeMillis();
    try {
        IProduct productType = PluginFactory.getInstance().getPlugin(
                "NOAA", "PLUGIN");

        productType.decode(fileName, msg);
        returnVal = productType.store();

    } catch (Exception e) {
        throw new EdexException("Unable to decode "+Util.prints...(this.fileName),e);
        //e.printStackTrace();
    } finally {

        long end = System.currentTimeMillis();
        this.execCount++;
        this.execTime += end - begin;
        logger.info("Ingesting message file: " + fileName + " in " +
                    + (double) ((double) end - begin) / 1000 + "s");
    }
    return returnVal;
}
```

Retrieving the message as a byte[ ]

Getting a reference to the Plugin

Decoding the file name to get metadata

Putting the store() on the Outbound Endpoint

Sending a message to the Outbound End Point

Wiring the SOA Service Into the Container

ingest.xml

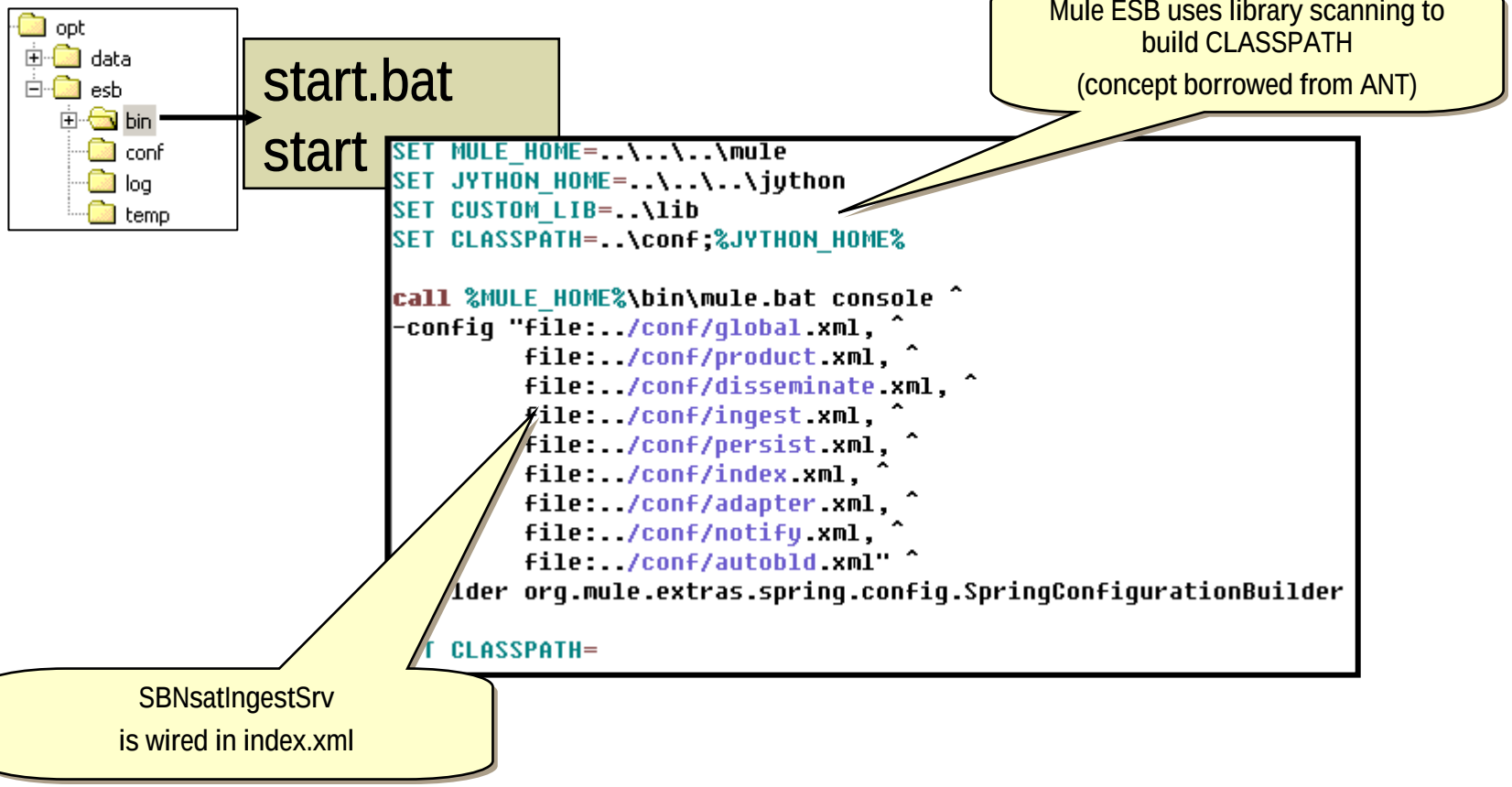
```
<mule-configuration version="1.0">

        <mule-descriptor name="SBN-Sat-Ingest" singleton="true"
                implementation="com.raytheon.edex.services.noaa.SBNsatIngestSrv"
                outboundEndpoint="vm://persistVMQueue">
                <inbound-router>
                        <endpoint
                                address="file:../../data/sbn/sat/?transformers=FileToString">
                                <properties>
                                        <proper    name="pollingFrequency" value="1000" />
                                </properties>
                        dpoint>
                </inbou    uter>
        </mule-descript
```

Class reference to
SOA Service implementation

Inbound Endpoint is a ESB
File sniffer with dir reference

Outbound Endpoint
Virtual memory queue
SEDA scaled

Mule ESB/ Container Startup Reads Wiring



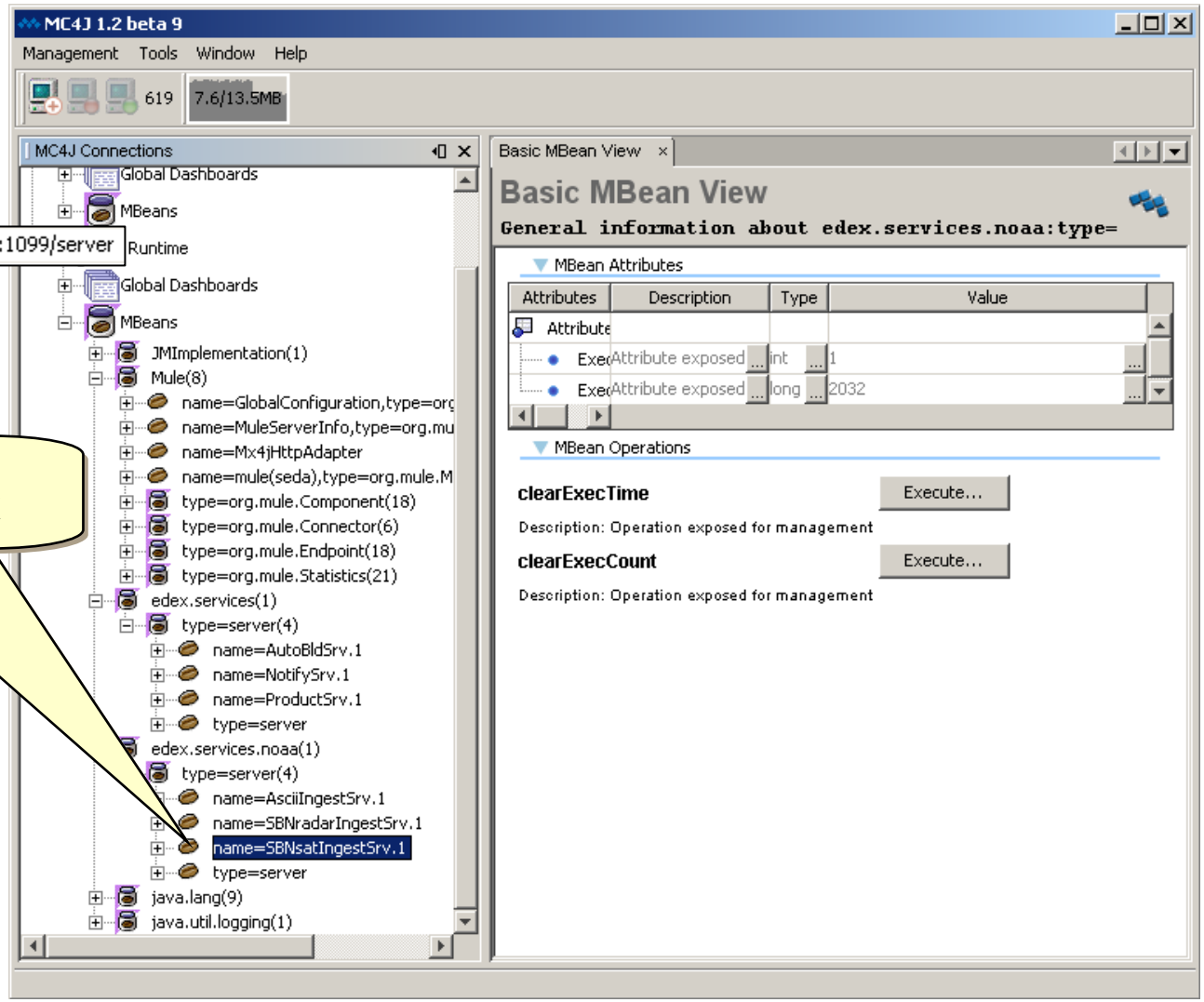
JMX Console Connects to Mule ESB Runtime
(JMX Is a T05 Capability, But T04 Has Some Basics)

Raytheon

Connect JMX to Mule
(activeMQ is automatic)

Server URL service:jmx:rmi:///jndi/rmi://localhost:1099/server

SBNsatIngestSrv
Remote Monitoring Through JMX



Exercise: Monitoring an SOA Service SBNsatIngestSRV Monitoring

Raytheon

Startup: Server Process {activeMQ, Mule, Tomcat}

1. Monitor Mule's log file.
2. Connect JMX console to Mule.
3. Note statistics with regard to SBNsatIngestSRV.
4. Ingest a raw IR satellite file.
5. Look at what happened to the log file and JMX console.

Extra Credit: Debug SOA Service

Same Exercise as Before but with Step-By-Step Debugging

1. Attach Eclipse IDE debugger to Mule.
2. Set breakpoint toward the beginning of the +process() method in SBNsatIngestSrv.
3. Ingest the raw IR satellite file.
4. Watch as the breakpoint is hit.
5. Step through the code from the breakpoint, watching the variables as they change.

Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 6: CAVE-Underlying Framework and Rendering (rev. 1)

February 14, 2008

AWP.TRG.SWCTR/TO6.ADE/CAVE06.01

Objectives

- General Introduction to CAVE
- Understand How CAVE Renders Geospatial, Vector, and x-y Data

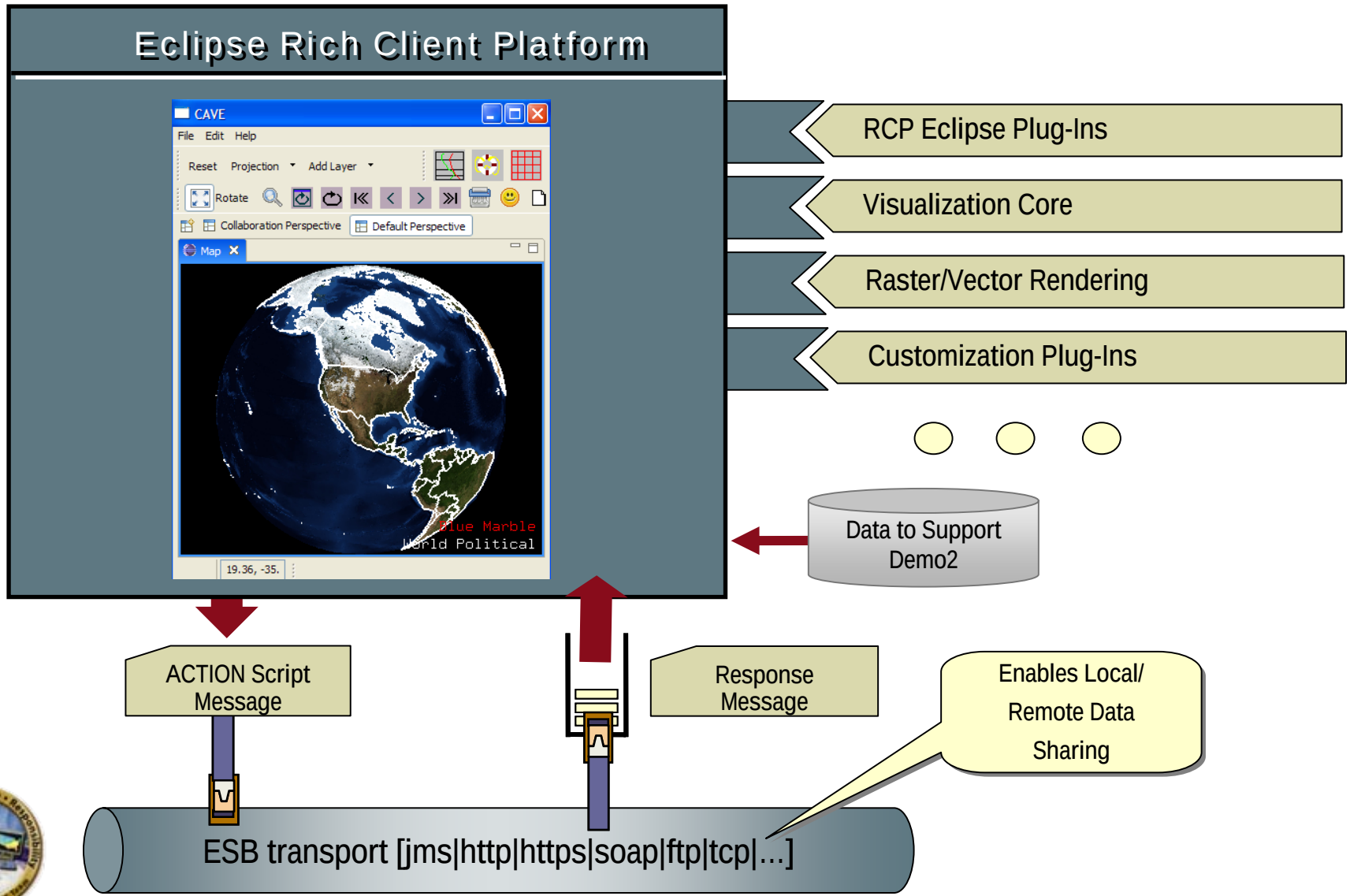
Motivation and Goals of CAVE

- Minimize GUI infrastructure, maximize reuse
- Performance
- Extendability

CAVE: Vision

- Bring together the visualization capabilities found in N-AWIPS, D2D, GFE, FX-C, FX-Net, and the Hydro GUIs in a common framework
- Maximize rendering and framework patterns reuse across GUI applications to minimize maintenance costs

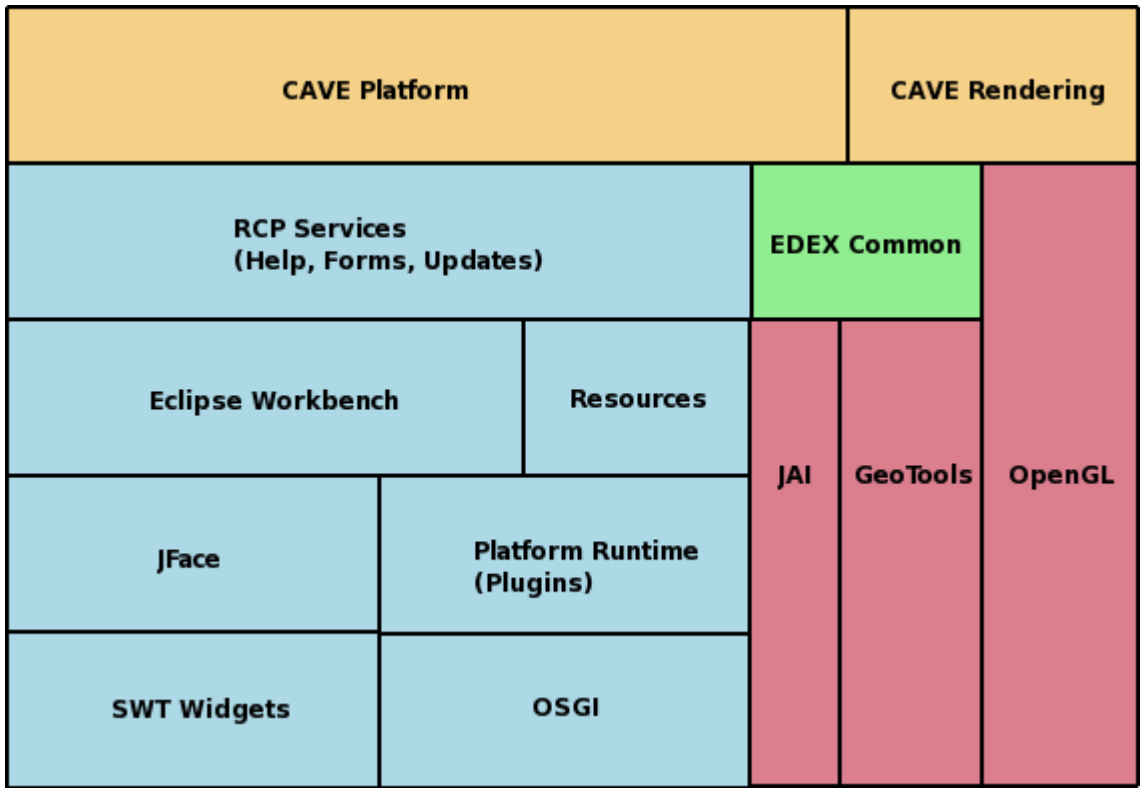
CAVE Top-Level Concept



Goal 1: Minimize Infrastructure, Maximize Reuse

- Developing boilerplate code: Waste of time and effort. Been done before, and probably better
- CAVE utilizes Eclipse Rich Client Platform for its infrastructure
  - Implemented a set of Plug-Ins for Eclipse Rich Client Platform
    - CAVE in ADE 1.0 has 400 Java Classes in approximately 50 Plug-Ins
- Other infrastructure:
  - GeoTools for geo-location
  - Mule for communication
  - Many other Open Source products (vecmath, units, etc.)

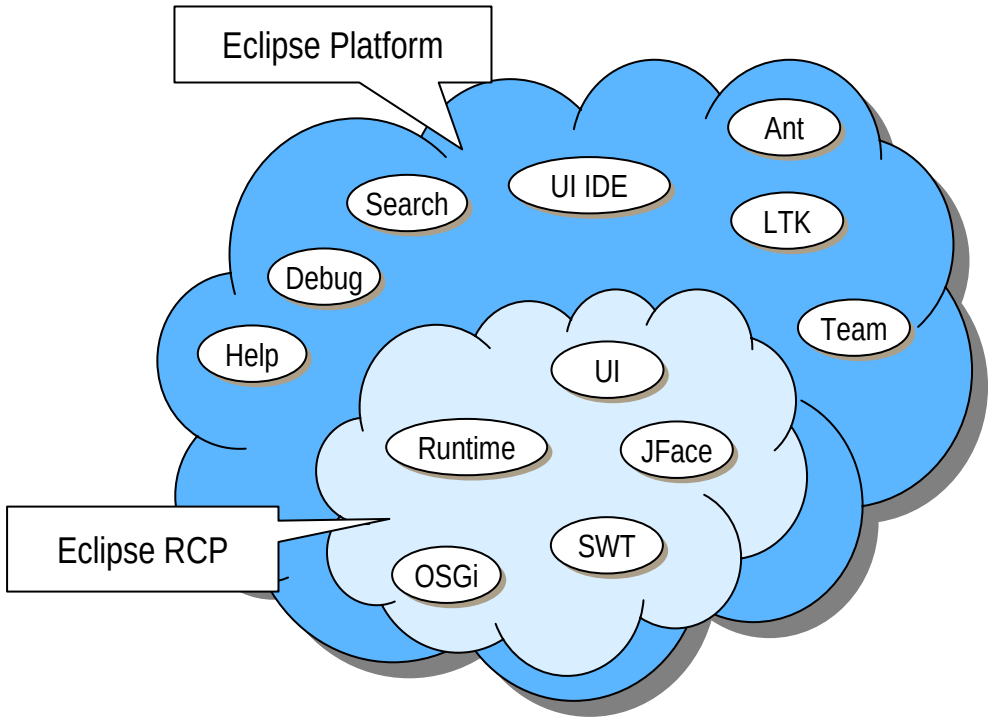
Goal 1: Minimize Infrastructure, Maximize Reuse
Architectural Diagram



Goal 1: Minimize Infrastructure, Maximize Reuse
Eclipse RCP

Raytheon

- Minimal set of Plug-Ins needed to build a rich client application collectively known as Rich Client Platform



Goal 1: Minimize Infrastructure, Maximize Reuse
Characteristics of Eclipse RCP Application

Raytheon

- Desktop Application: A Thick Client, not a Web browser application
- Runs on multiple platforms using native widgets
  - Looks like a Windows App on Windows, looks like a Linux GTK app on Linux, etc.
- Rich UI with consistent metaphor
  - Operates like modern applications with which users are familiar
  - Tight integration with desktop OS
  - Supports “drag and drop,” printing, etc.
- Easy deployment
  - All platforms can be built simultaneously
  - Installation usually no more than copying a folder

Goal 1: Minimize Infrastructure, Maximize Reuse
Eclipse Technologies Used in CAVE

Raytheon

- OSGi and Runtime: Provides plug-in model
- UI
  - SWT (Standard Widget Toolkit)
    - Platform-independent native widget toolkit
  - JFace (Framework providing higher-level UI abstractions)
    - Menu bar, tool bar, content area, status line, viewers, actions, …
  - Workbench, text, forms, GEF available
- Help and User Assistance Mechanisms
  - Help (html/xml based, context sensitive, search), Intro, Cheat Sheets
- Deployment (Update Manager)
  - APIs to programmatically update
- Runtime Extension / Extension Point Model
  - plugin.xml

BREAK


Innovation • Partnership • Responsibility
The Raytheon Warrior Team

Goal 2: Performance

- Extremely Important: Performance-driven systems in place today. Replacement technologies should also be performance driven
- CAVE Performance Approach:
  - Fully harness the power of current- and future-generation graphics cards using OpenGL. (Today's graphics cards are several orders of magnitude faster than CPU at many operations.)
  - Use advanced caching and data decimation techniques to make rendering of large data usable
  - Make the application as multi-threaded as possible so that the user is not actively blocked while waiting for tasks to complete

Goal 2: Performance
Performance Using Raster Data

Raytheon

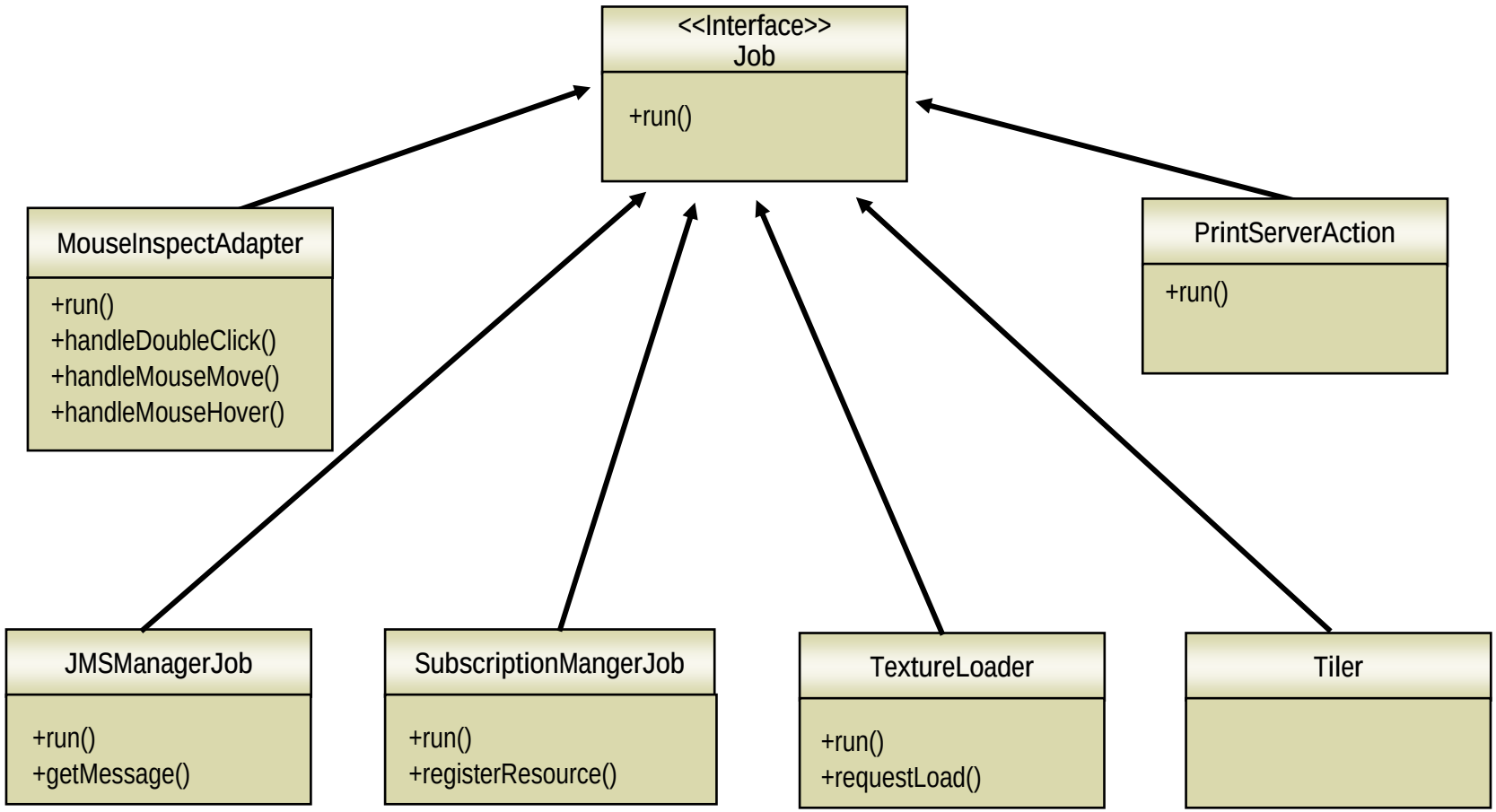
- Quad-Tree Tiling implemented in HDF5 for Large Raster Data
  - Raster data “pre-staged” into tiled levels
    - Coarsest: Level 0
    - Least Coarse: level n
    - Each level twice the resolution of the previous level
    - Tiles can be any size (although experience shows that 256 x 256 tends to yield good performance)
  - Tiles automatically brought in as needed
    - Tiles only brought in at the zoom level at which they are applicable
    - Tiles only brought in when they are over a spatial area currently being viewed
  - Tiles retained in memory until resource no longer needed, or space allocated for tile cache is exhausted
    - Tiles evicted using a “Least Recently Used (LRU)” algorithm
    - Two levels of storage: Graphics Card and Memory
  - *Demo: Tile Loading and Eviction*
  - *Demo: Tile Format on File-system*

Goal 2: Performance
Performance in Vector and Plot Data

Raytheon

- Making Vector Rendering Fast
  - Use OpenGL concepts for high-speed vector drawing
  - Use automatic vector decimation and other algorithms to reduce level of detail automatically
  - Allows for rendering vector data an order of magnitude larger than many systems
  - *Demo: Vector Decimation*
- Performance of Plot Data
  - Generate plot data asynchronously, requesting as the user zooms in, rendering the individual plot offline, and bringing it into the display as it is available
  - *Demo: Plot Rendering*

Goal 2: Performance
Performance of UI – Eclipse Jobs Enable Multiprocessing



Goal 3: Extendability

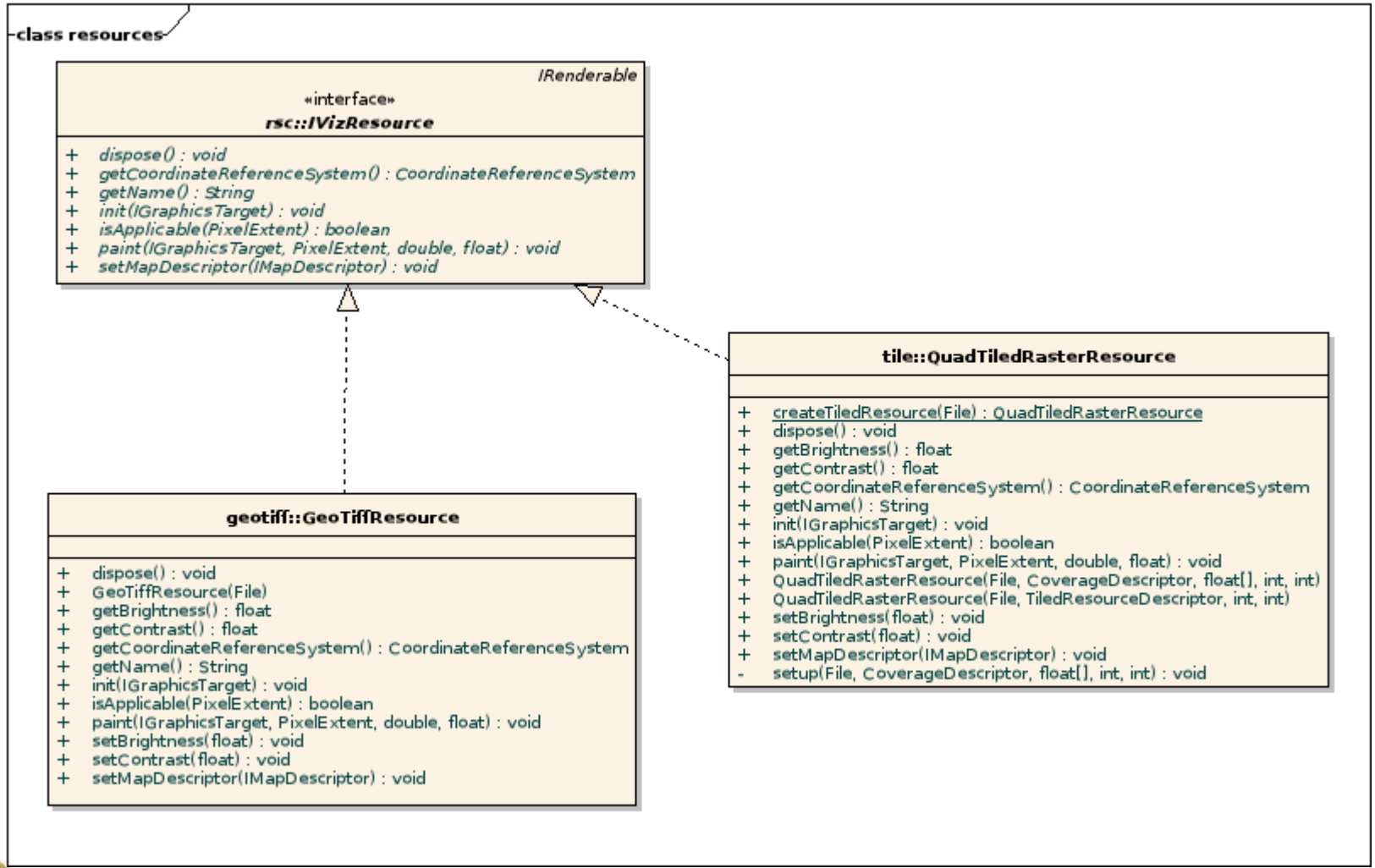
- Allow CAVE to be extended easily by outside users
- As much as possible, allow users to extend by writing their own Plug-Ins − not by modifying framework
  - Accomplished by using Eclipse Extension Points
- Example: Defining a new Data Type with a registered file extension (e.g., “tif”) for TIFF files
  - User creates a new Plug-In that defines a resource, and adds a Plug-In descriptor that registers the “.tif” file extension to their resource
  - No modification to core code

BREAK

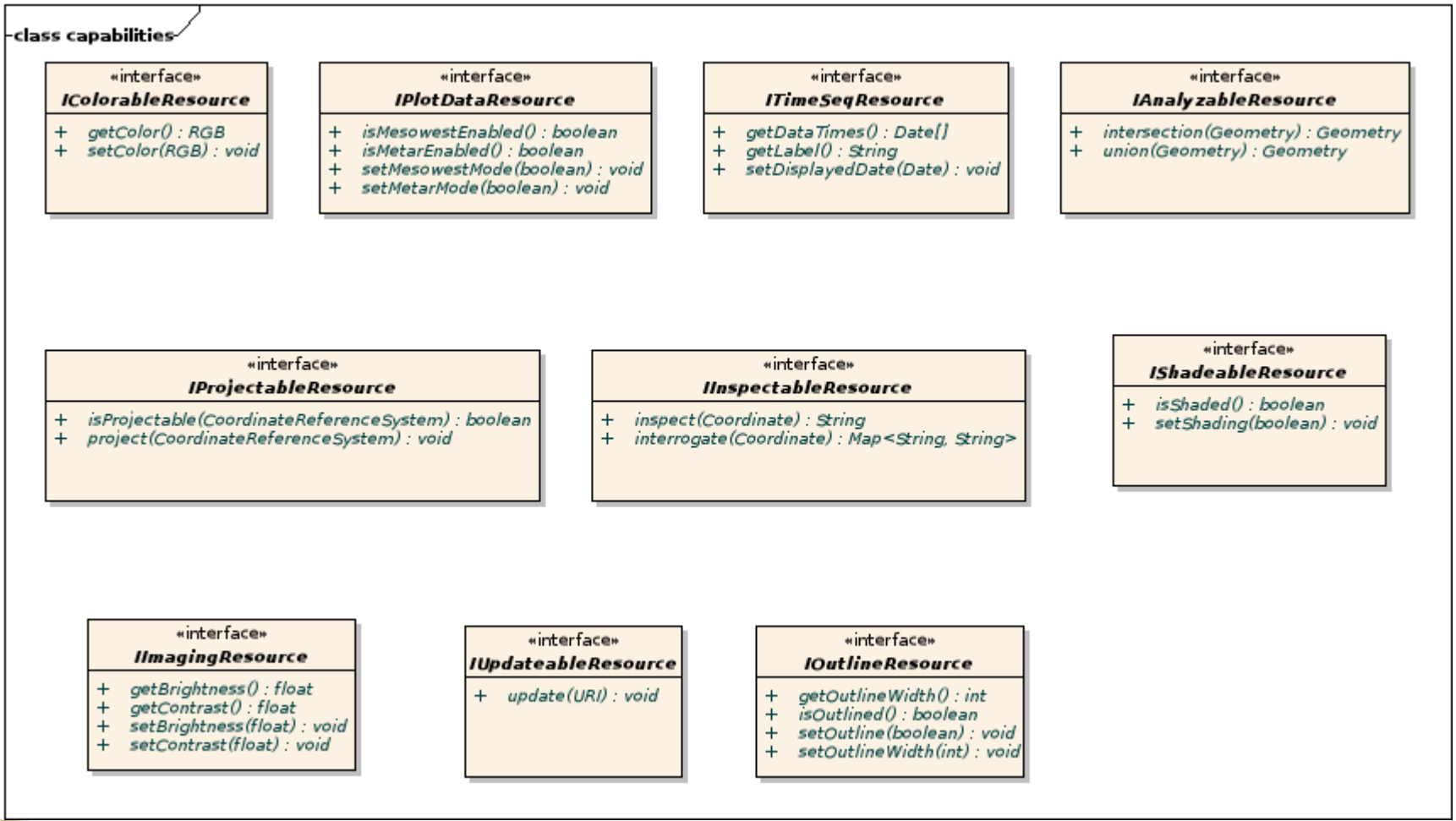
CAVE Core Data Structure Concepts

- Examine a Few of CAVE's Core Data Structures
- Resources, Capabilities, and Map Descriptors
  - Resources: Describe a “Layer” on the Map
  - Capabilities: Interfaces implemented by Resources that provide a capability
    - Example: **IColorableResource** is implemented for resources that can have its color changed
  - Map Descriptors
    - Contain a set of Resources and properties about the display

CAVE Resource Structure – ADE 0.1 Similar in Concept to D2D Depictable



CAVE Resource Capabilities



CAVE Resource Capabilities

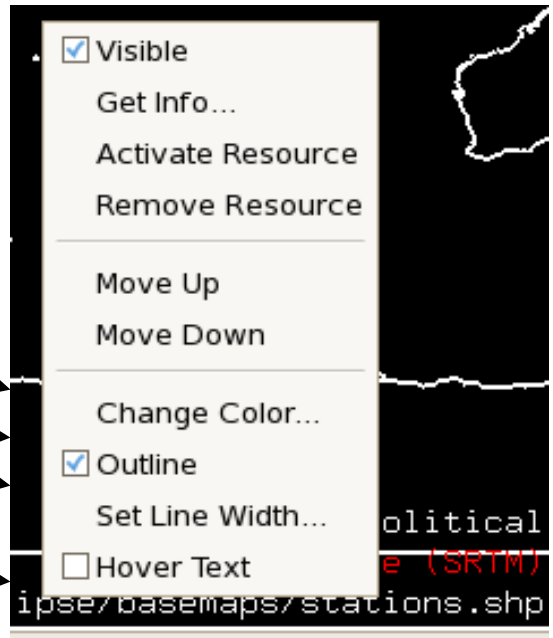
- Many (not all) capabilities exposed to the user through the contextual menu of the legend automatically – if the Resource implements the interface

- Example:

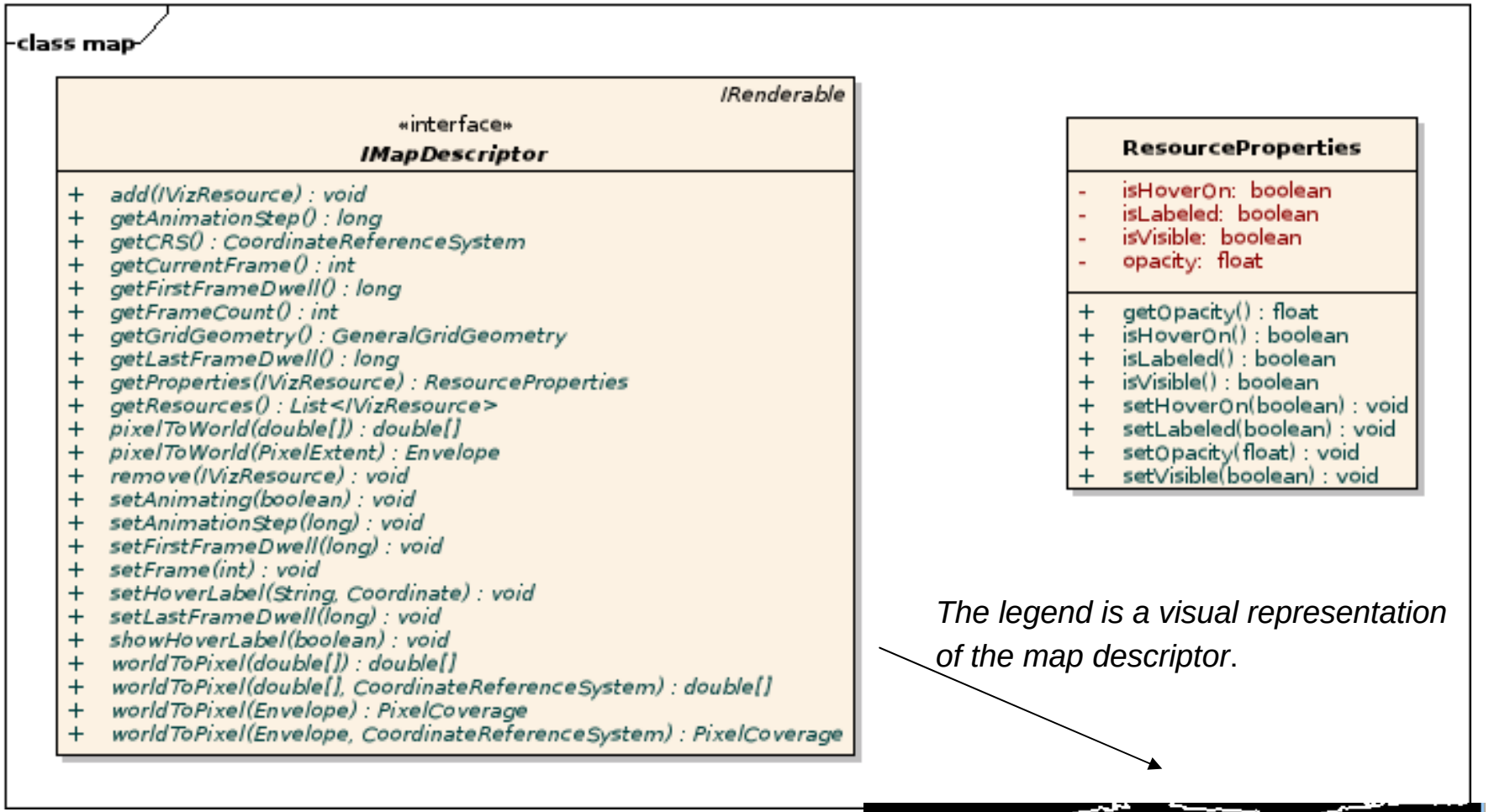
IColorableResource

IOutlineResource

IInspectableResource



Map Descriptor



The legend is a visual representation of the map descriptor.

Exercise: Hands-On Rendering CAVE

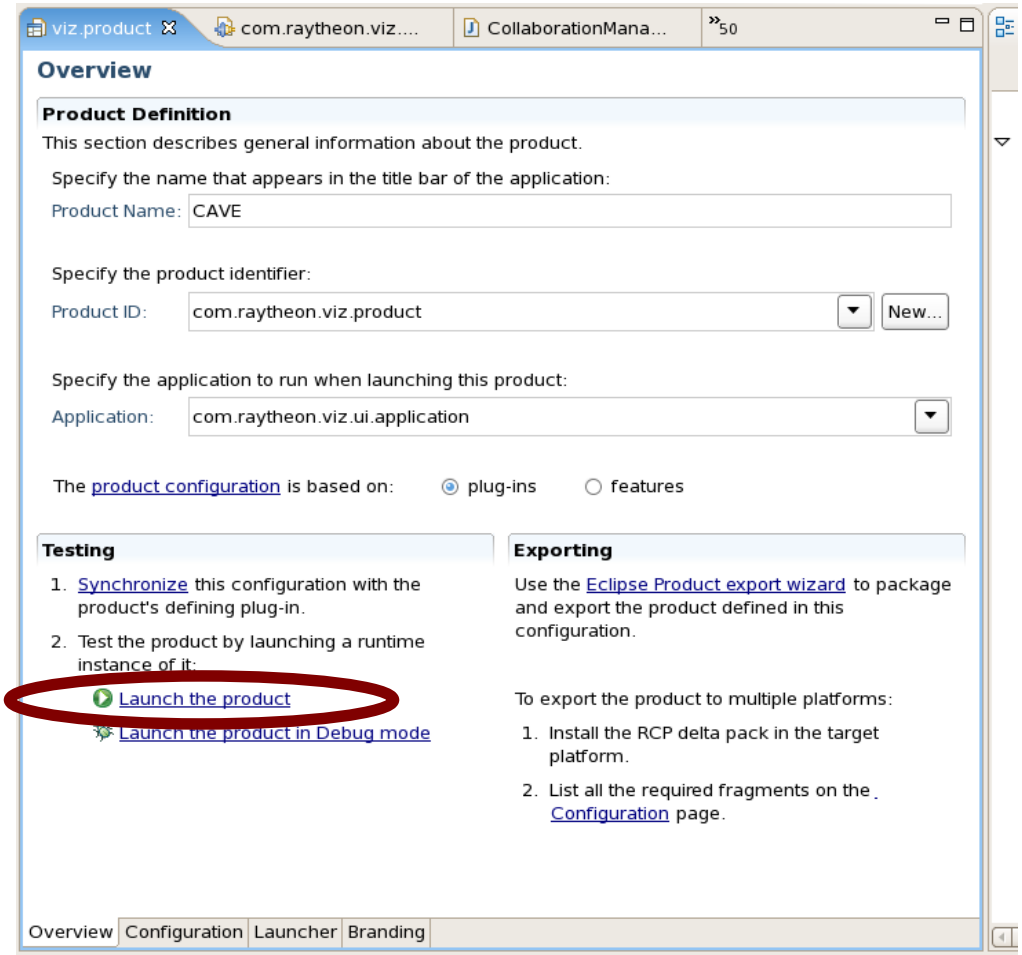
- Practical Exercise:
  - Launching CAVE From Source Baseline
  - Imaging Resources in CAVE

Exercise:
Launching CAVE From the Baseline

1. From inside Eclipse, locate the “com.raytheon.viz” project
2. Expand the project (click the triangle)
3. Double click on viz.product

Exercise:
Launching CAVE From the Baseline (Cont'd)

4. After the description page loads, click on the "Launch the Product" blue hyperlink.



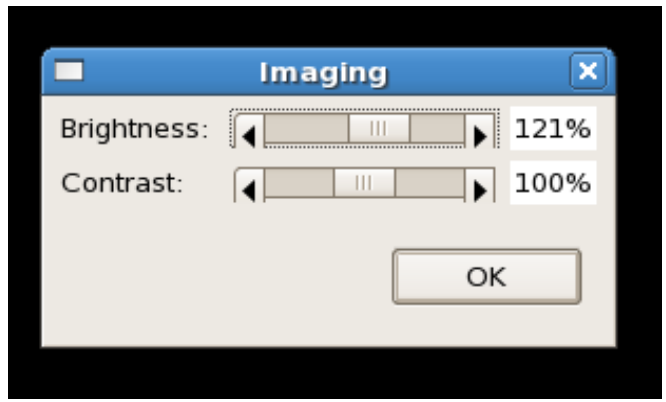
Exercise:
Imaging Resources in CAVE

Raytheon

Before modifying code, we will learn how to open up and manipulate a few resources in CAVE.

Demo/Exercise: CAVE GeoTIFF Demo

1. *Open CAVE*
2. *File->Open GeoTIFF*
3. *Open "test.tif"*
4. *Right click on Legend and experiment with the capabilities: Brightness (inside Imaging), Contrast (inside Imaging), and Visibility.*



LUNCH


Innovation • Partnership • Responsibility
The Raytheon Team's Team

Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 7: CAVE-User Interface (rev. 1)

February 14, 2008

AWP.TRG.SWCTR/TO6.ADE/CAVE07.01

Objectives

- CAVE baseline orientation
- Add functionality by modifying plugin.xml
- Add a new menu item and custom resource

Baseline Orientation

- 50 Plug-Ins in present CAVE environment
  - Each with its own project in Eclipse
  - Each built independently (although some Plug-Ins depend on others)
- Some important CAVE Plug-Ins:
  - Core Plug-In: Provides rendering and infrastructure
  - User interface Plug-In: Provides core user interface capability
  - Drawing & collaboration CAVE Plug-In: Provides drawing and collaboration support
  - Several Open Source Java projects that have been repackaged as Plug-Ins (org.jibx, org.geotools, etc.)

Important Capability Plug-In Packages

Plug-In	Description
com.raytheon.viz.geotiff	Support for the GeoTIFF geospatial imagery standard
com.raytheon.viz.shapefile	Support for the Shapefile geospatial vector standard
com.raytheon.viz.pointdata	Support for pointdata plotting
com.raytheon.viz.volumebrowser	Support for the volume browser UI
com.raytheon.viz.grid	Gridded data Rendering
com.raytheon.viz.radar	Radar Rendering
com.raytheon.viz.satellite	Satellite Rendering
com.raytheon.viz.ui.tools*	Provides map interactions

Open Source Repackaged Plug-Ins

Plug-In	Description
ncsa.hdf5	Provides HDF5 Capabilities
net.sf.ehcache	LRU Caching support
org.apache.activemq	JMS messaging support
org.apache.batik –	SVG rendering
org.apache.commons*	various Apache utilities
javax.units	Unit Conversions
javax.vecmath	Vector Math
javax.media.opengl	OpenGL Support
org.geotools	Geotools Geospatial Library
org.jivesoftware.smack	XMPP Client for Collaboration
ncsa.hdf5 –	Provides HDF5 Capabilities

Eclipse Plug-In XML

An example of plugin.xml (from core):

```
 <extension
         point="com.raytheon.viz.core.resource">
      <resource
            class="com.raytheon.viz.core.rsc.shp.ShapefileResource"

factoryClass="com.raytheon.viz.core.rsc.shp.ShapefileFactoryAdapter"
            name="ShapefileResource">
         <fileType
               fileExtension="shp"
               name="Shapefile"/>
      </resource>
</extension>
```

This extension registers the Shapefile type to the ShapefileResource (and the .shp file extension).

Eclipse Plug-In XML (cont'd)

Another example from Drawing's plugin.xml:

```
<action
     class="com.raytheon.viz.drawing.WeatherSymbolTool"
     icon="icons/thunderstorm.gif"
     id="com.raytheon.viz.drawing.WeatherSymbolTool:17"
     label="Thunderstorm"
     state="false"
     style="radio"
     toolbarPath="drawing/g1"
     tooltip="Thunderstorm"/>
```

This code adds the Weather Symbol tool to the toolbar as the Thunderstorm tool. Note the ":17" at the end. This corresponds to the 17.svg in basemaps.

BREAK

Exercise: Plug-In XML

Objective: Add an entry to plug-in xml to add another symbol

1. Modify plugin.xml by adding another <action> block (copy the thunderstorm block).
2. Point to testSymbol.svg and testSymbol.gif instead of 17.svg and 17.gif.
3. Give it a meaningful label and tooltip.

Exercise:
Creating a Custom Imaging Resource

Raytheon

Based on some of the capabilities we've experimented with in CAVE, we'll add a new item to the “Add Layer” menu containing our GeoTIFF resource, with custom brightness and contrast settings.

1. Save a bundle using the “Save Bundle” item in the file menu.
2. Open the bundle XML using your favorite text editor
3. Clear the screen
4. Load the modified bundle
5. Verify the bundle matches what you expect

Exercise:
Creating a Custom Imaging Resource (Cont'd)

Raytheon

6. Once you have a bundle that you like:
   - Copy the bundle file to cave/etc/staticMenu/Demos.
   - Restart CAVE. The menu item should show up in the Add Layer

References: Eclipse

- Eclipse RCP home page
  - http://eclipse.org/rcp
  - http://eclipse.org/community/rcp.html
  - News://news.eclipse.org/eclipse.platform.rcp
- Books on Eclipse
  - *Eclipse Rich Client Platform* by Jeff McAffer
  - *SWT: The Standard Widget Toolkit, Vol. 1* by Northover

BREAK


Innovation • Partnership • Responsibility
The Raytheon Warrior Team

Additional Information

- Books
  - *Patterns of Enterprise Application Architecture:* Fowler
  - *Enterprise Integration:* Patterns Hohpe
  - *Lucene in Action:* Gospodnetic
  - *Hibernate in Action:* Bauer
  - *Enterprise Service Bus:* Chappell {Sonic ESB slant}
  - *SVG for Web Designers:* Jason…
  - *SVG Essentials:* Eisenberg
  - *Spring in Action:* Walls
  - *Lighter, Faster, Java* by Spring inventor
  - *Eclipse* (extending and writing plug-ins …)
  - *Xdoclet in Action:* Walls
  - *ANT* – developers handbook
  - *Junit in Action:* Massol
  - *Java 2D Graphics:* Knudsen

Additional Information (cont'd)

- Links
  - Mule ESB + SPRING: http://mule.codehaus.org/
  - Subversion CM: http://subversion.tigris.org
  - ECLIPSE IDE framework & plug-ins: http://www.eclipse.org
  - ActiveMQ JMS broker: http://www.activemq.org
  - PostgreSQL RDBMS: http://www.postgresql.org
  - JBossCache: http://www.jboss.org/products/jbosscache
  - RHINO JS scripting: http://www.mozilla.org/rhino
  - MINA: http://directory.apache.org/subprojects/network/index.html
  - Batik SVG tools: http://xml.apache.org/batik
  - Hibernate relational to object mapping: http://www.hibernate.org

Additional Information (cont'd)

- Wx-related projects
  - Unidata NetCDF
  - VisAD
  - IDV
  - OpenGIS/GeoTools

Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 8: CAVE Visualization Plug-Ins (rev. 1)

February 14, 2008

AWP.TRG.SWCTR/TO6.ADE/CAVE08.01

Prerequisites/Objectives

- Prerequisites
  - Familiarity with CAVE baseline (TO4 Training)
  - Familiarity with Java and Eclipse
  - Exploration of the CAVE source code baseline
  - ADE 1.0 installed

- Objectives
  - Understand the mechanisms required to extend CAVE
  - Write a new Plug-In to extend CAVE functionality

  Estimated Time: 2 hours

Introduction to Extending CAVE

CAVE Extension Mechanisms

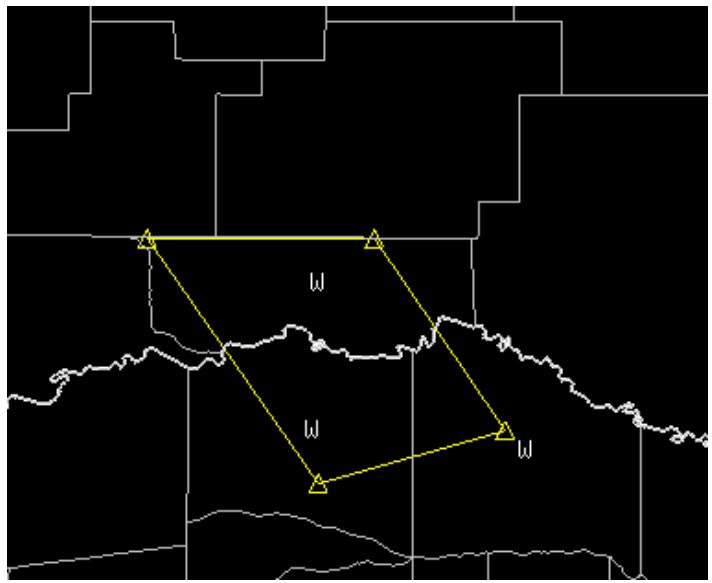
- Mechanisms for extending CAVE
  - New Resource
    - Provides the ability to create a new renderable layer or data type
  - New Toolbar item
    - Facilitates launching an action or activating a modal tool
  - New Menu item
    - Facilitates launching an action
  - New Editor
    - Provides a mechanism for creating custom interactions in the main pane

Extension Mechanism: New Resource

- A new resource that provides
  - The ability to visualize a new datatype
  - The ability to draw to screen (see AcetateLayer)
    - Drawing resources
    - Visualizations for interactions

Resource Extension Example: WarnGen

- WarnGen
  - Actually contains two of our extension types
    - A new layer and
    - An action that is attached to a toolbar button
  - Layer provides a way to render the manipulated warning/watch on the map:

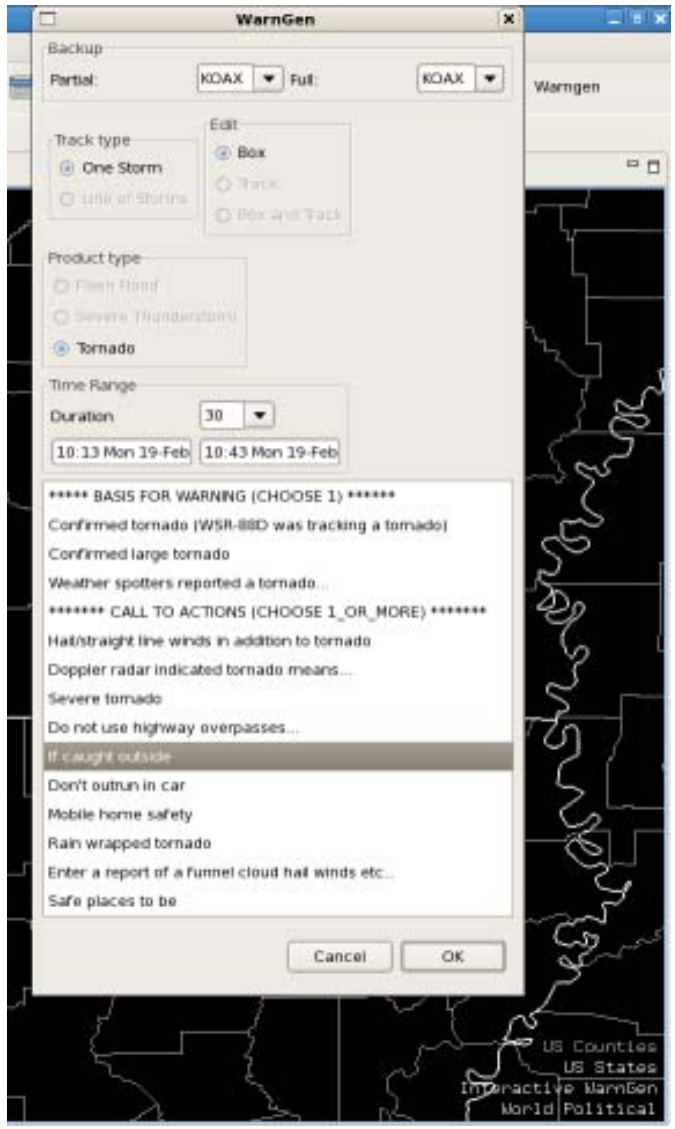


Extension Mechanism: Toolbar Item

- Toolbar item
  - Extends AbstractTool
  - Allows any sort of action to run
    - Mouse interaction
      - ► By enabling a mouse handler
    - Example: PanTool
    - Pop-up window
      - ► By creating a Dialog
      - ► Example: Collaboration Login
    - Floating Palette
      - ► By creating a Modeless dialog
      - ► Example: WarnGen
    - Custom code
  - Possible to run as a modal (state-based) or non-modal
    - Modal: Pan Tool . . . Non-Modal: Loop Preferences Dialog
    - To run modal, toolbar item should be set as style “toggle” and extend abstract class com.raytheon.viz.ui.tool.AbstractModalTool

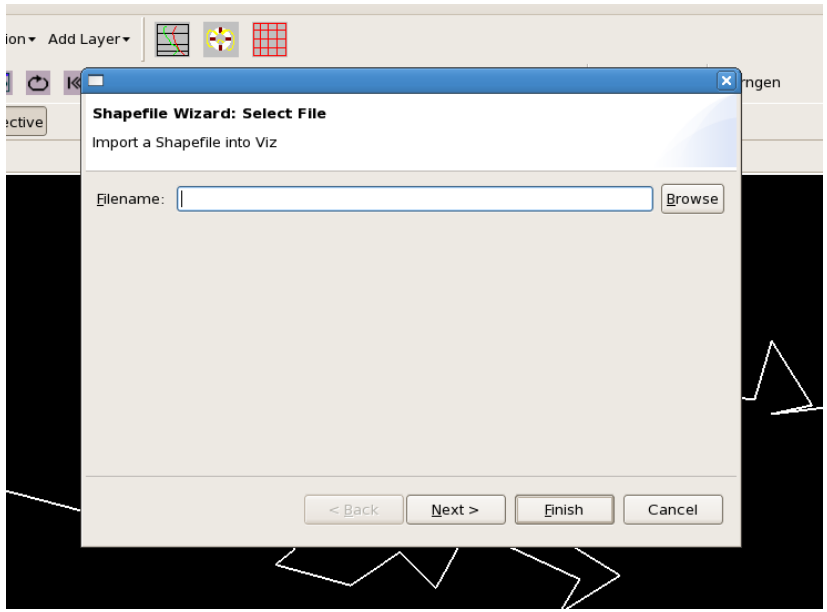
Toolbar Extension Example: WarnGen

- WarnGen toolbar action
  - Complicated
  - Actually provides multiple types of interactions, e.g.:
    - Enables a mouse handler that allows the user to draw and manipulate warnings on the map. The handler interacts with the warning layer, which actually draws the warning
    - Displays a pop-up window that displays the warning parameters.



Extension Mechanism: Menu Items

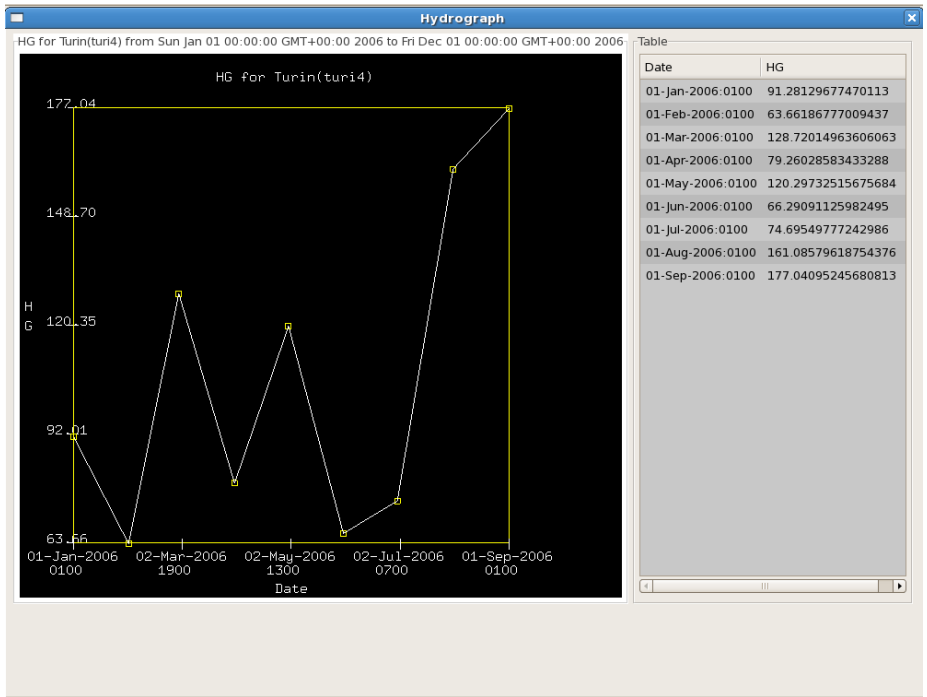
- Nearly identical to toolbar
  - Example: Open Shapefile
    - Opens a dialog box (actually a Wizard)



Note: *In existing CAVE baseline, menus are contributed through Java Code, not through Eclipse extensions. This will change during future TOs to allow more flexibility.*

Extension Mechanism: New Editor

- Used to display radically different kind of data (non map-based)



BREAK

Exercise: Extending CAVE

Creating a New Plug-In

In this example, we create a Plug-In that extends CAVE in the two most common ways (a toolbar action and a resource).

- First, we create a simple renderable resource that displays a rectangle on the screen over a predefined area.
- Then, we create an example toolbar action that adds the new layer to the map.

Getting Started

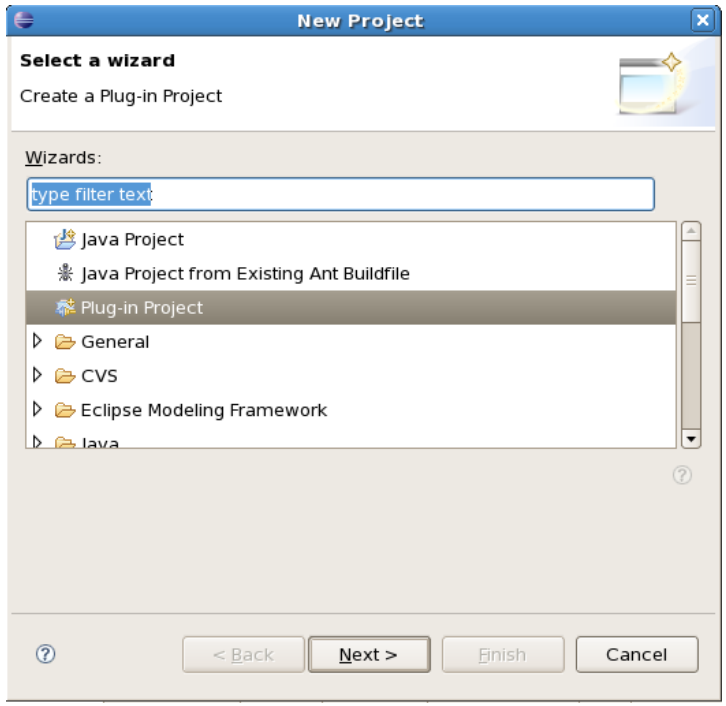
- Creating a new project
- Setting up environment options
- Adding your Plug-In to the build distribution

Creating a New Project in Eclipse

1. From the File menu, choose New->Project.

2. Choose “Plug-in Project.”

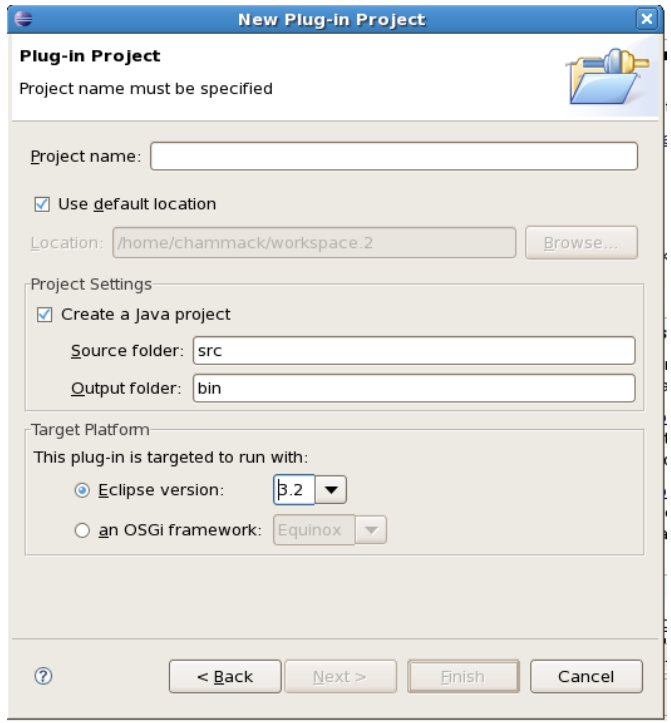
3. Click “Next >.”



Creating a New Project in Eclipse (cont'd)

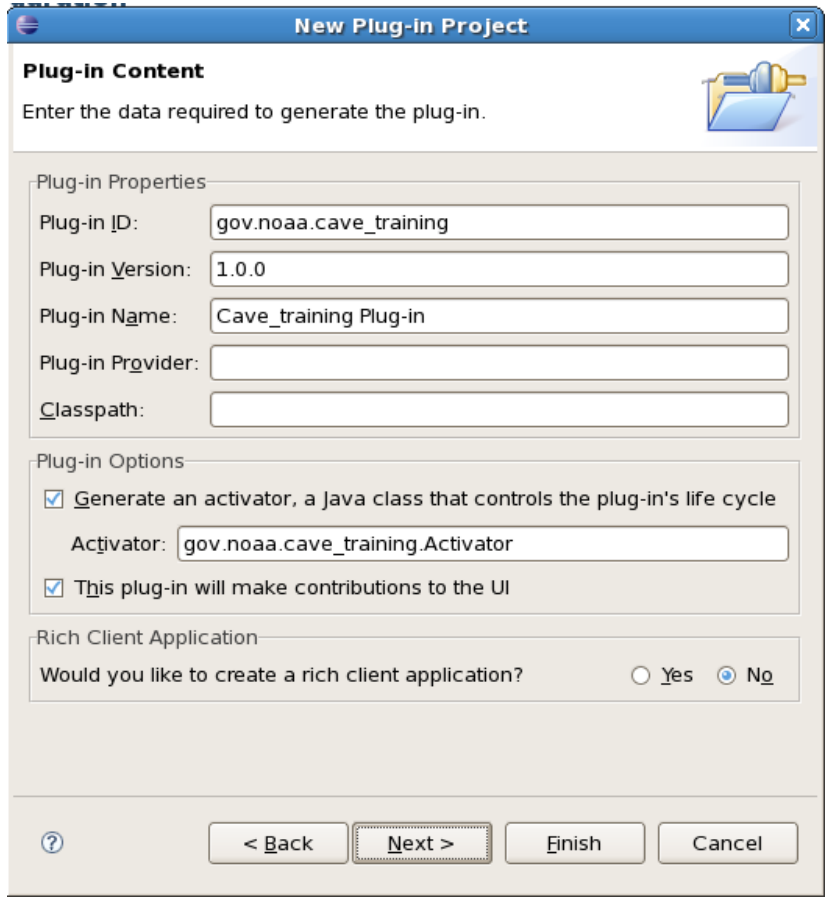
4. Set Project name to: gov.noaa.cave_training.

5. Click "Next>."



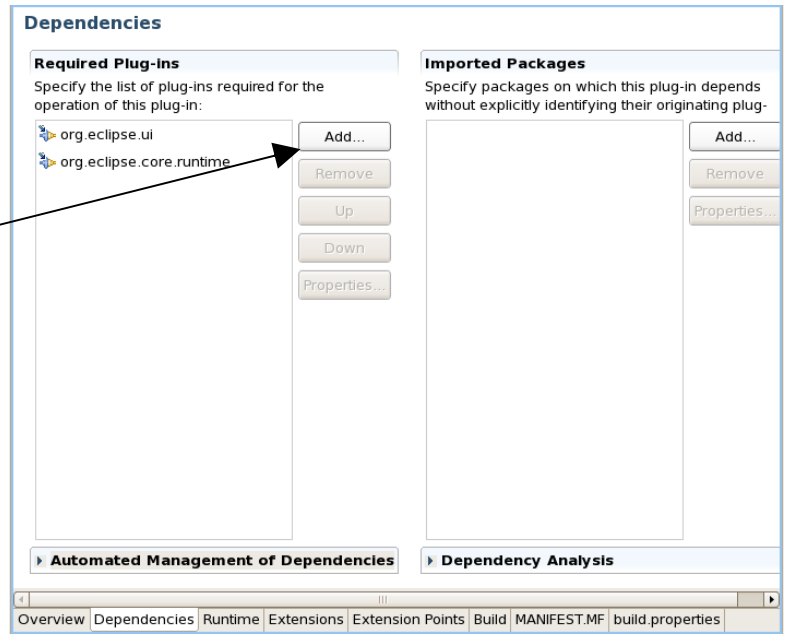
Creating a New Project in Eclipse (cont'd)

6. Click "Finish."



Setting up the Project Environment

1. From the main view in Eclipse, choose the recently created project. Expand it, and the META-INF directory.
2. Double click on the MANIFEST.MF file. Choose the "Dependencies" tab. The following screen appears:



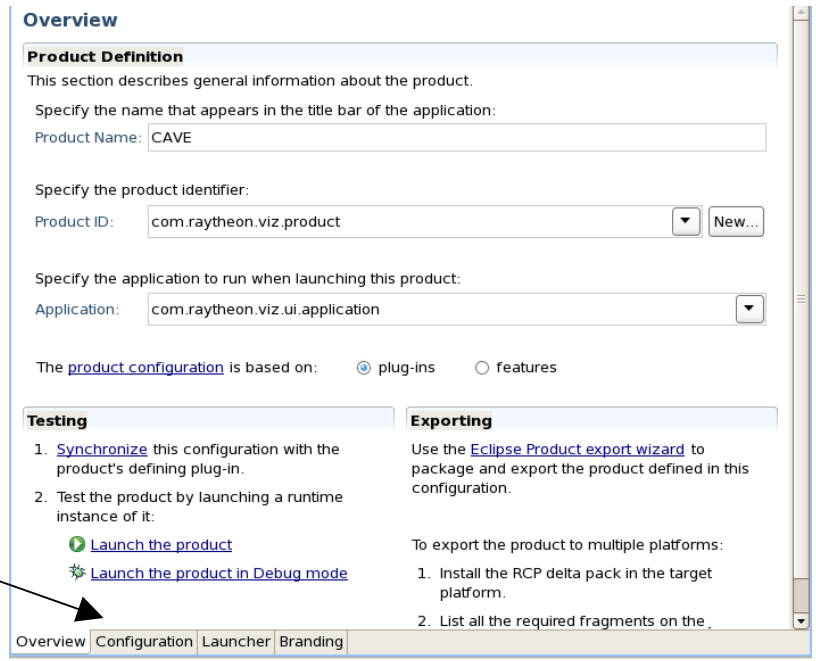
3. Add
   - com.raytheon.viz.core
   - com.raytheon.viz.libs
   - com.raytheon.viz.ui

Adding to the Project Build

Now, add the new Plug-In to the CAVE product so that is included in the build.

1. Inside of the com.raytheon.viz plugin, find “viz.product” and double click on it. The following screen appears:

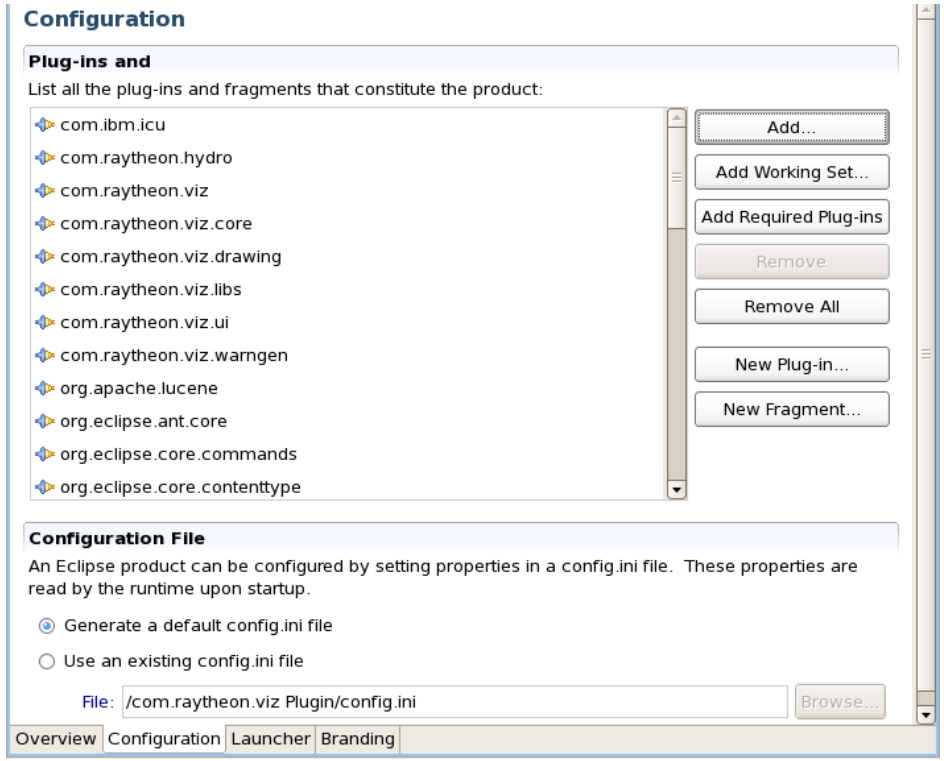
2. Choose “Configuration.”



Adding to the Project Build (cont'd)

1. Click "Add."

2. Add the gov.noaa.cave_training Plug-In.



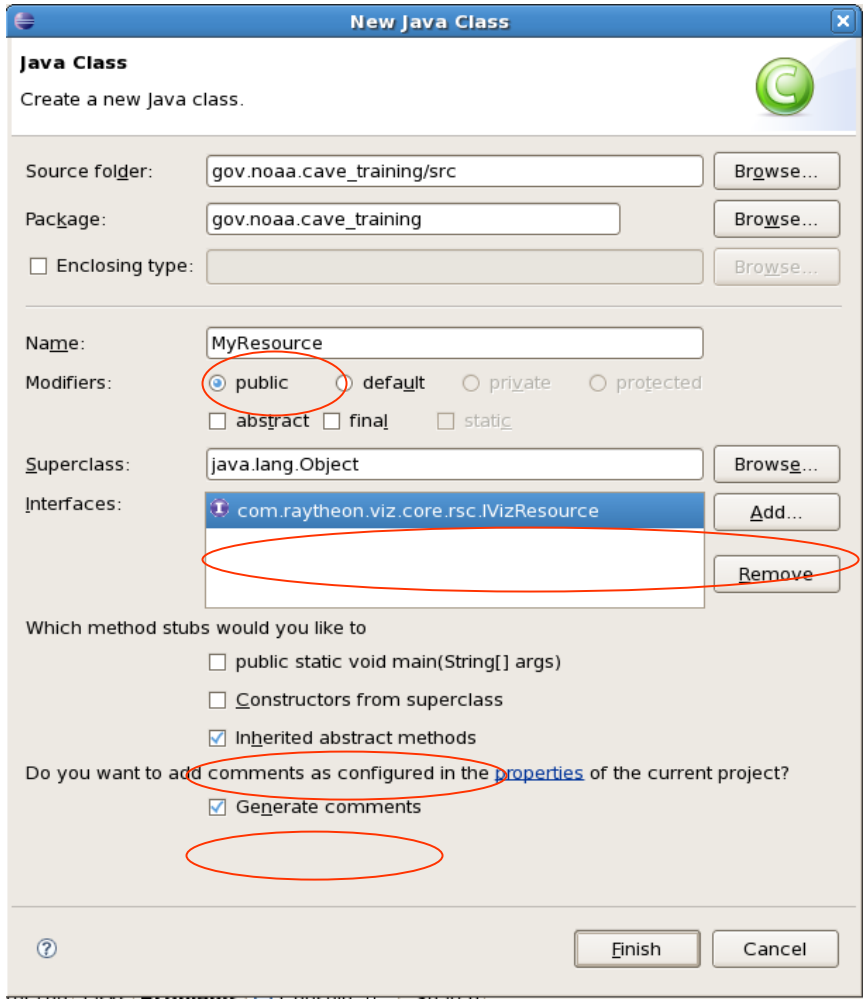
Creating a New Resource

1. Create a new class:
   - gov.noaa.cave_training.MyResource.
   - From inside the gov.noaa.cave_training project, expand the “src” grouping and select the gov.noaa.cave_training package.
   - Right click on the package and choose New->Class.

Creating a New Resource (cont'd)

2. Fill out the class template as shown in this example.

3. Click Finish.



Creating a New Resource (cont'd)

- Use Eclipse as an aid in filling out the method headers:
  - We only need to implement a few select methods:
    - getName (determines the name on the legend):

```
public String getName() {
   return "My Resource";
}
```

    - isApplicable (hint that determines if rsc should be drawn):

```
public boolean isApplicable(PixelExtent extent) {
   return true;
}
```

Creating a New Resource (cont'd)

- paint(): (using methods in the IGraphicsTarget passed in as an argument to the paint method, draw to the screen):

```
public void paint(IGraphicsTarget target, PixelExtent extent,
  double zoomLevel, float alpha) throws VizException {
  target.drawLine(0, 0, 1000, 1000, new RGB(255,0,0), 1.0f);
}
```

– In this example our resource simply draws a line from screen coordinate (0,0) to screen coordinate (1000, 1000) using the color red, with a 1 point line width.

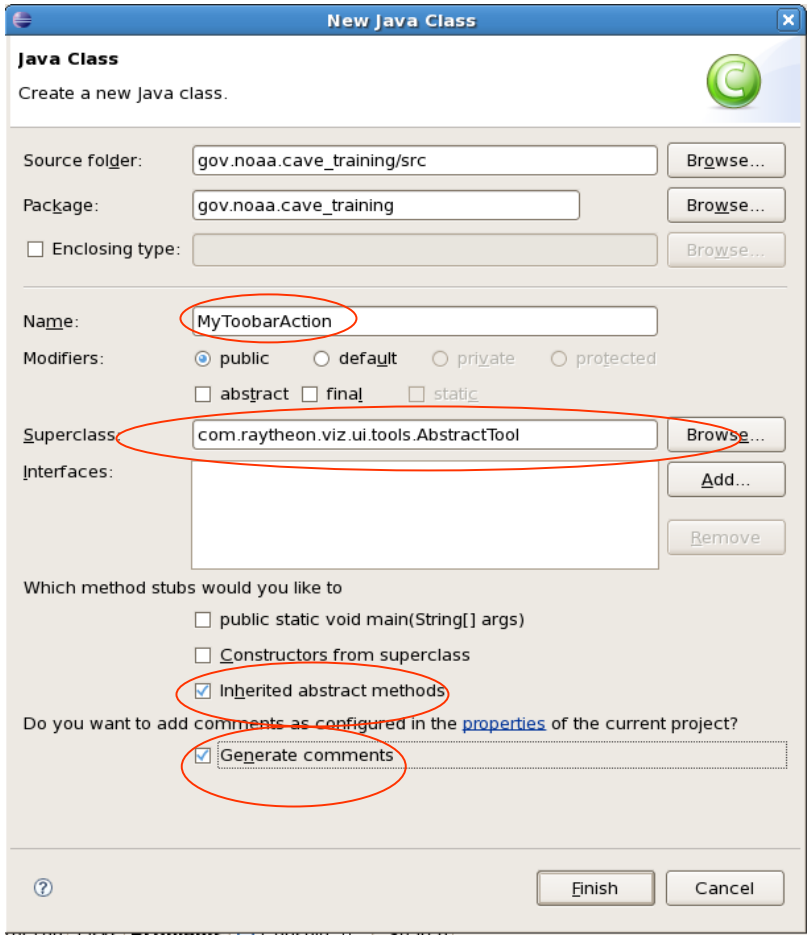
– In real resources, the IMapDescriptor is used to translate map/resource coordinates into screen coordinates.

Creating a New Resource (cont'd)

- We've now created a resource, but this type will not be utilized until an action is created to instantiate it.
  - There is no hook for CAVE to ever create our resource.

Add Toolbar Item Code

1. Create another new class:



Add Toolbar Item Code (cont'd)

2. Fill out the method runTool:

```
protected void runTool() {

  try {
     MyResource myResource = new MyResource();

     AbstractEditor editor = ((AbstractEditor) VizApp.getCurrentEditor());

     editor.getDescriptor().add(myResource);

     editor.refresh();
  } catch (WrongProjectionException e) {
     e.printStackTrace();
  }
}
```

Note: *This code instantiates our resource, adds it to the list of map resources, and refreshes the screen.*

Adding a Toolbar Action

We have our action class, but it will never be called unless Eclipse is made aware of it.

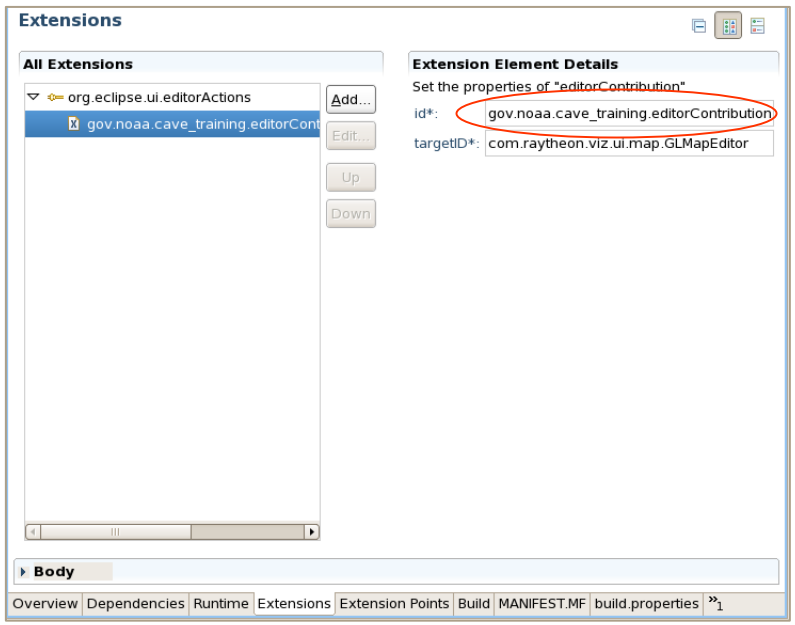
- We will notify Eclipse to expose our Action in the form of a toolbar item
- We do this by adding XML to our plugin.xml in our Plug-In

1. Double click on MANIFEST.MF in our project.

2. Choose the Extensions tab.

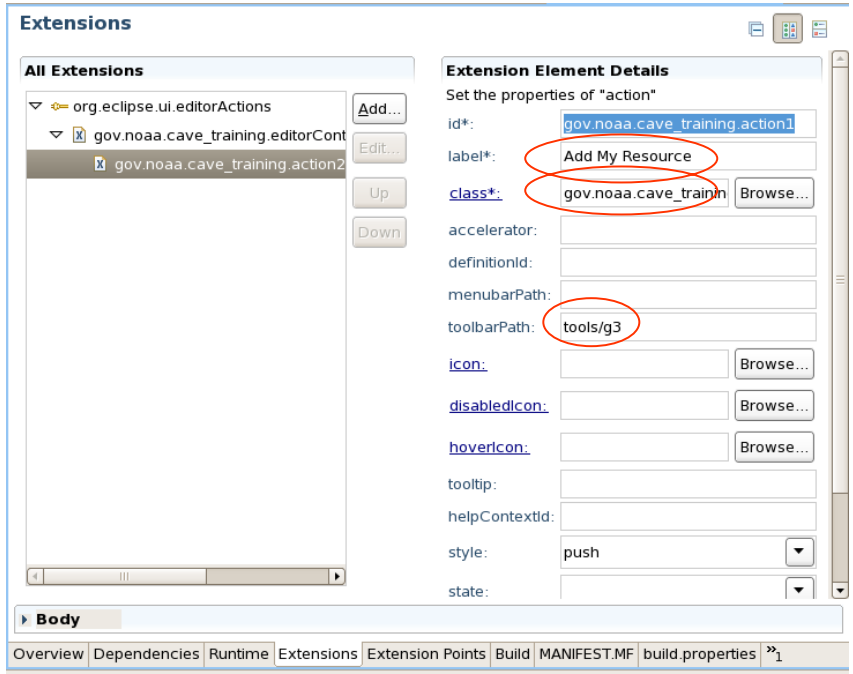
Adding a Toolbar Action (cont'd)

3. Create a new extension that extends "org.eclipse.ui.editorActions" by clicking on Add.
4. It should activate against the editor "com.raytheon.viz.ui.map.GLMapEditor." (The code we wrote was designed to interact with it.)



Adding a Toolbar Action (cont'd)

5. Add the action by right clicking on the editorAction and choosing New->action.
6. Fill in: name, class.
7. Set toolbarPath to "tools/g3." ***Note:*** *This is a grouping placeholder. Tools specifies the main toolbar, g3 specifies the third grouping (from left to right).*



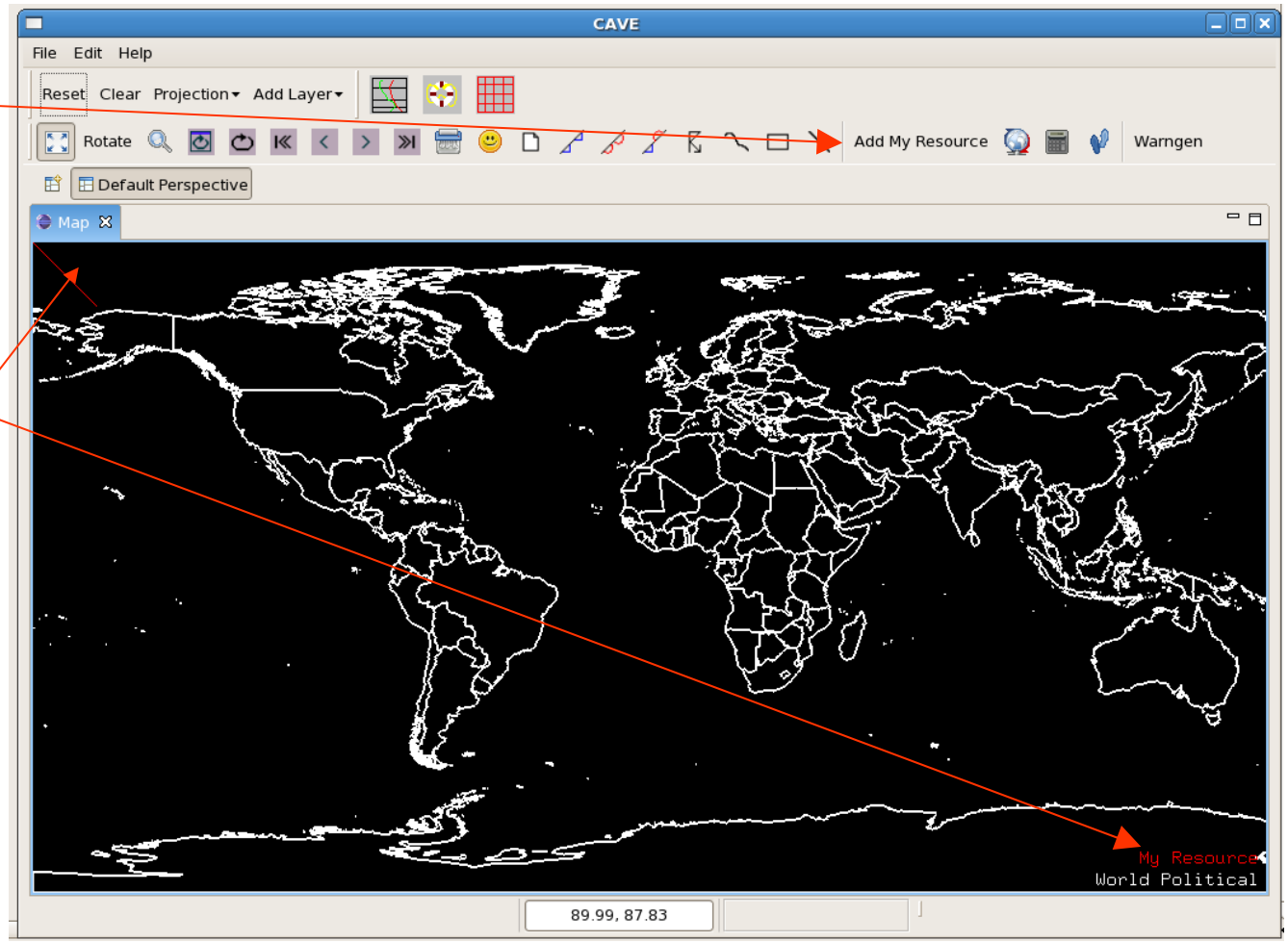
Run Our Example

1. From the viz base plugin, find viz.product.
2. Choose “Launch the product” from the Overview tab.

We should now have an action on the toolbar that we can click.

Run Our Example (cont'd)

- Our toolbar button
- Action when button is clicked



Wrap-Up

Summary

- Covered the basics on how to create a simple resource and action to extend CAVE
- Learned that adding menu items is extremely similar to adding a toolbar
- Extending new Editors for certain types of data is possible; nevertheless, developers are encouraged to use the built-in editors as much as possible

Resources

- On the ADE 1.0 DVD
  - Current code available for examination in the CAVE baseline
  - JavaDoc documentation available

Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 9: Installation/Deployment (rev. 1)

February 14, 2008

AWP.TRG.SWCTR/TO6.ADE/CAVE09.01

Prerequisites/Objectives

- Prerequisites
  - Root access. Only necessary if installing to a directory other than the user's home directory
  - VMWare Player installed on target machine. Only necessary if the VMWare image will be used, instead of installing EDEX.
- Objectives
  - Install the EDEX Services and CAVE Application to a Supported Platform

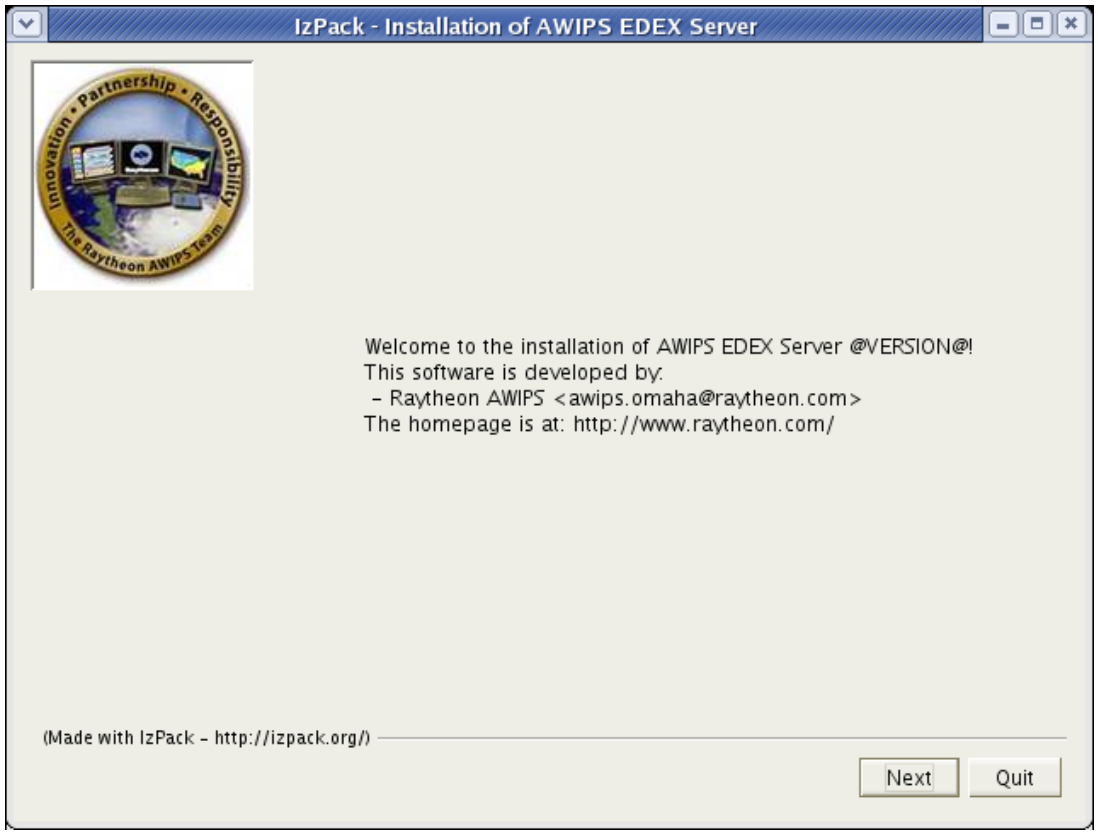
Estimated Time: 1 hour

EDEX/Cave Installation

- CAVE application and EDEX services installed using two separate installers
  - Each installer can install all files to the user's home directory. Installation can occur in any directory if root access is available to the installer.
  - EDEX can be run as a VMWare image. This image is a set of files and is available in the distribution. This option is available for use on Windows; CAVE can interact with a running VMWare instance.

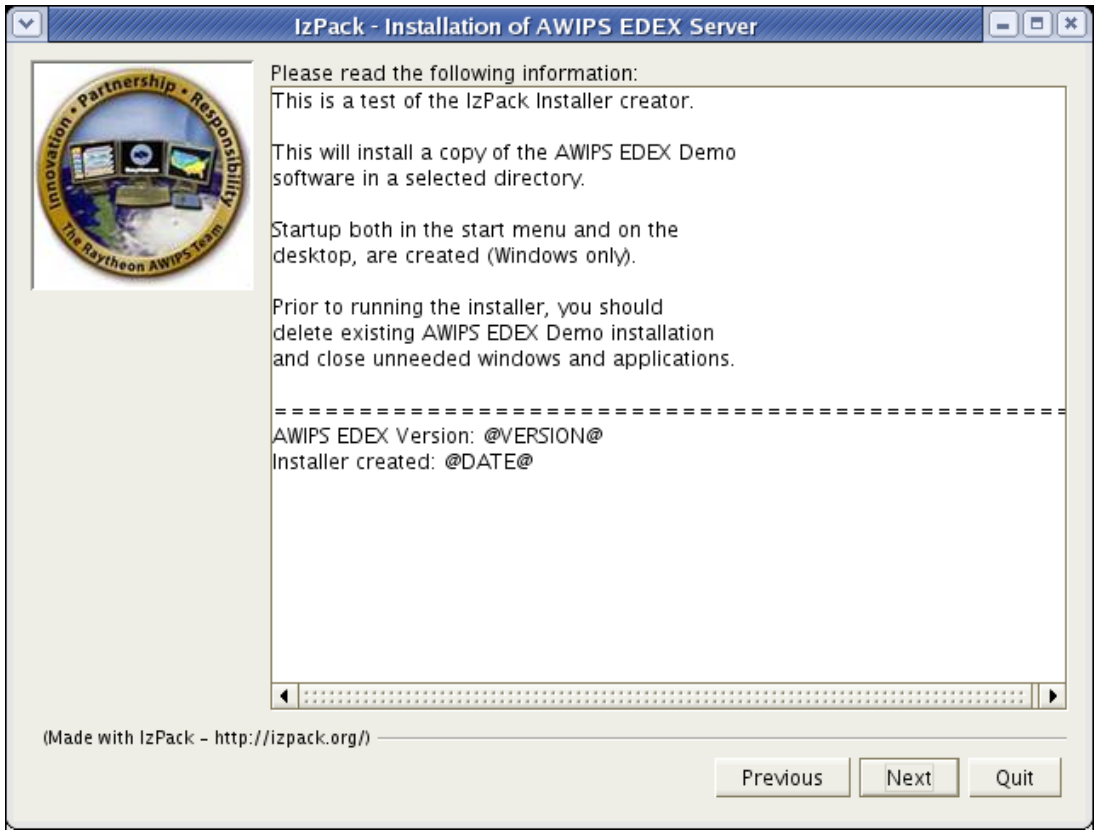
EDEX Installation (Linux)

- EDEX installer available for use on RHEL 4.2
  - On the start screen, select Next to proceed



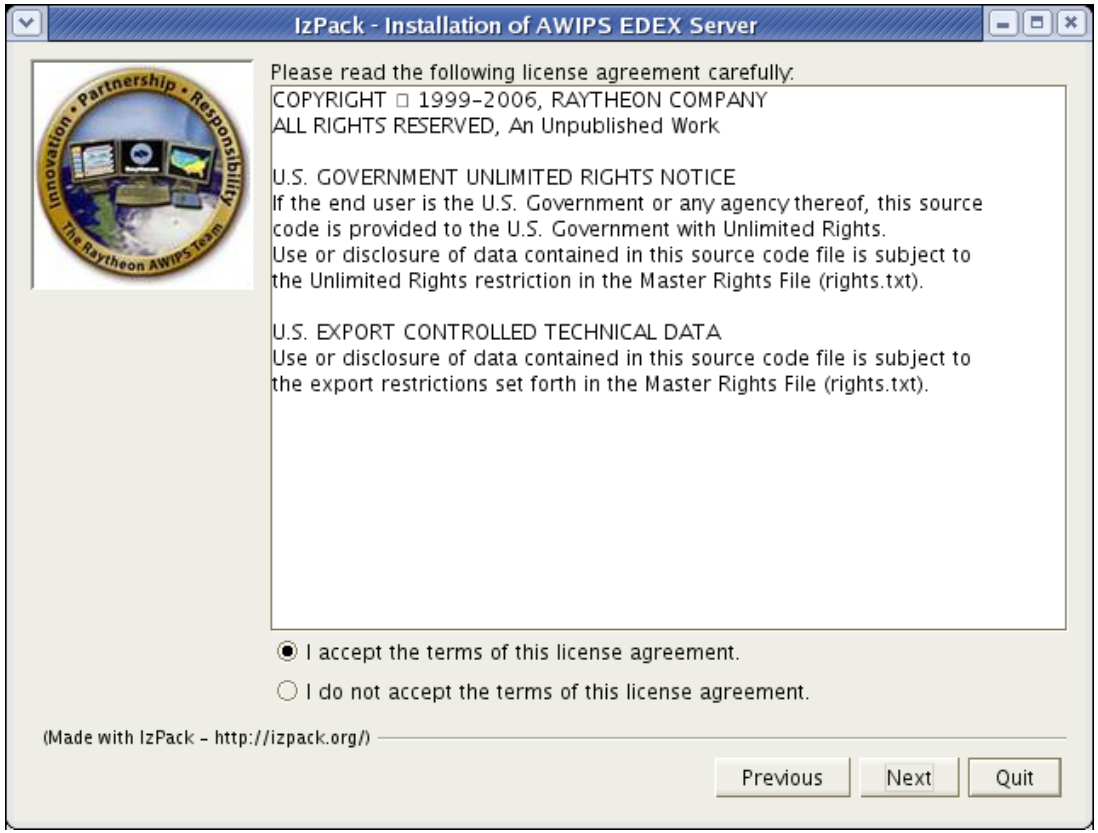
EDEX Installation (Linux) (cont'd)

– Select Next on the information screen



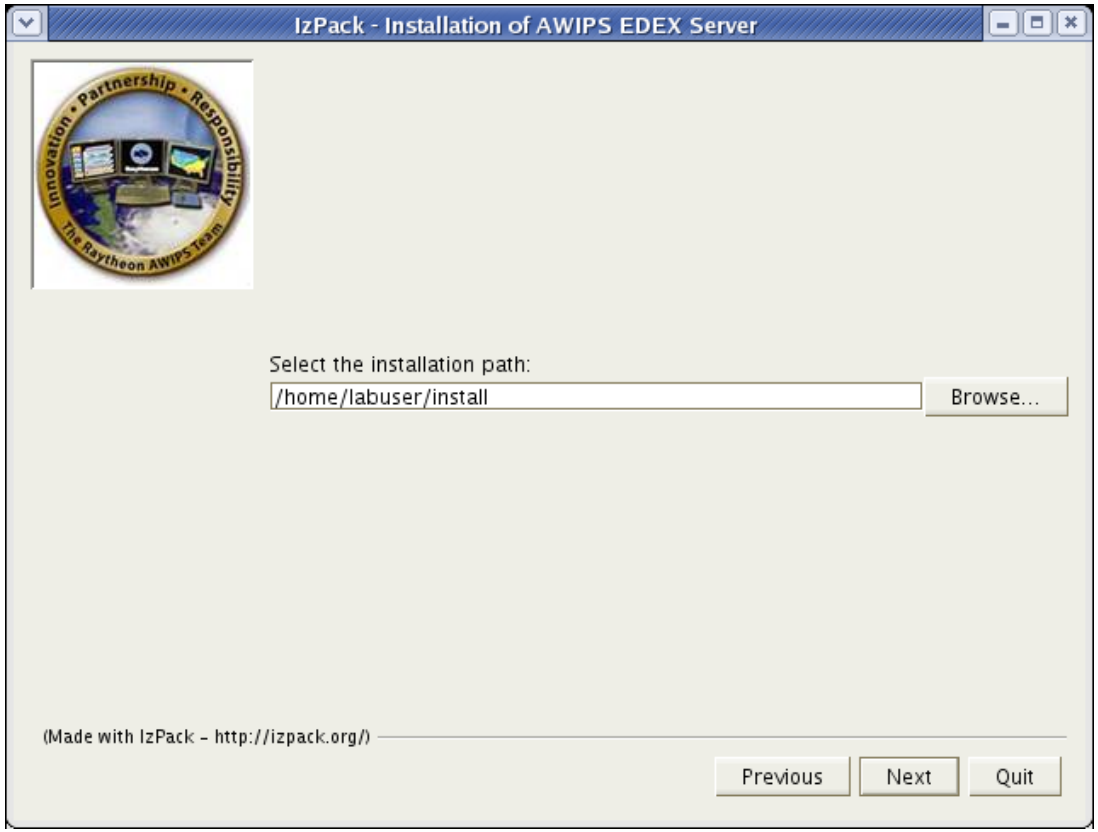
EDEX Installation (Linux) (cont'd)

– Accept the license agreement and select Next



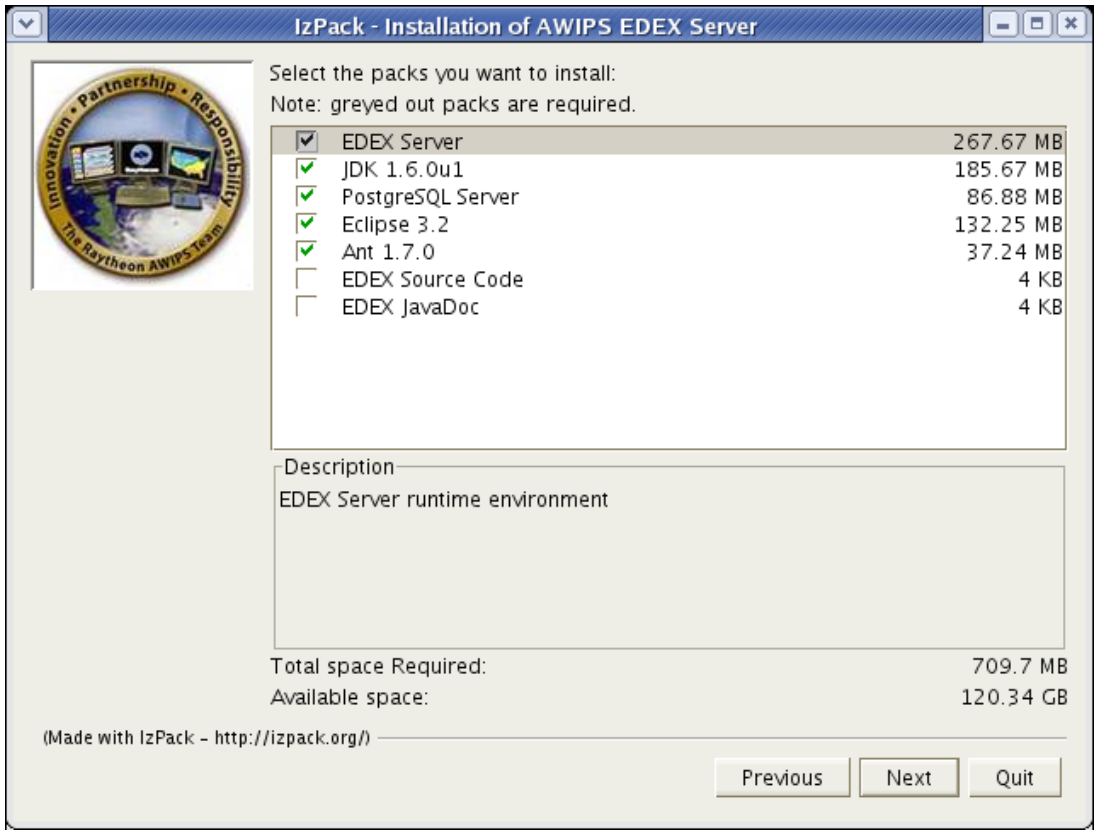
EDEX Installation (Linux) (cont'd)

– Enter the installation path (or browse to it) and select Next



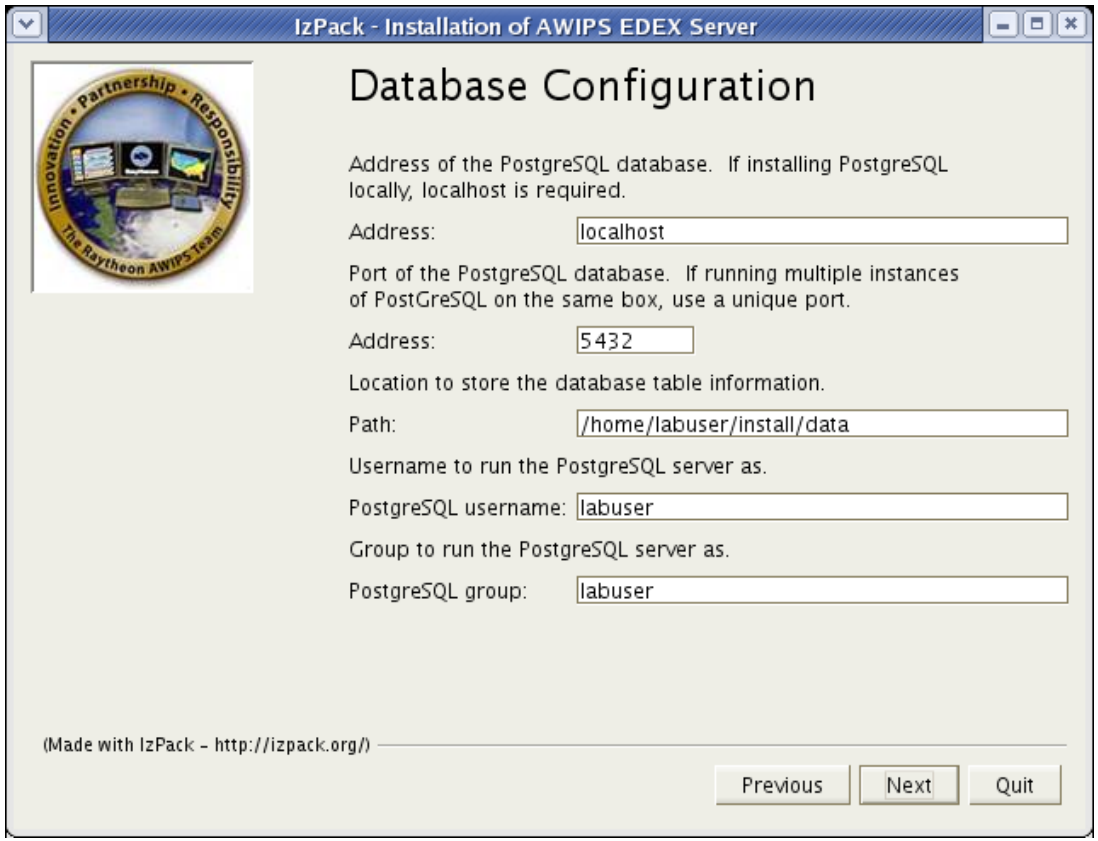
EDEX Installation (Linux) (cont'd)

- Select which components to install for the ADE
- Only the EDEX Server is required. Select Next to continue



EDEX Installation (Linux) (cont'd)

– Set the address of the PostgreSQL install, the port number to listen on, the path to store the PostgreSQL tablespaces and user/group information for the non-privileged user

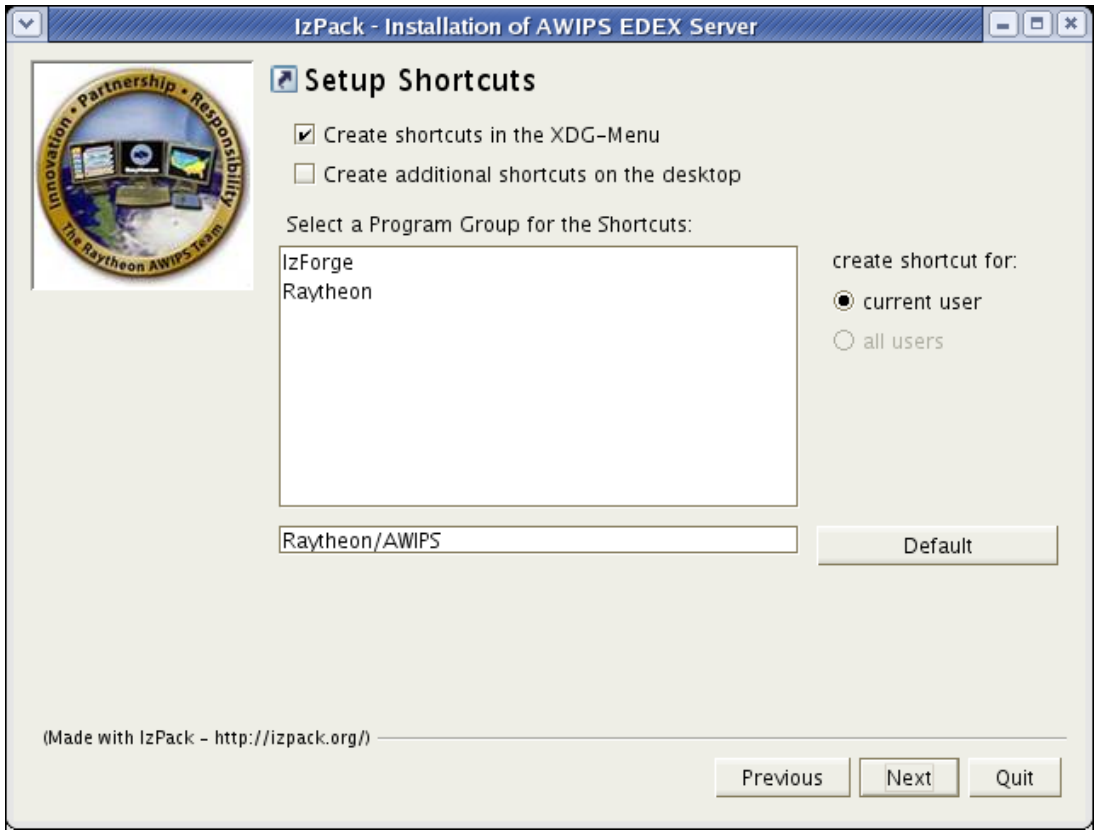


EDEX Installation (Linux) (cont'd)

– Enter the directory to store the HDF5 data and select Next

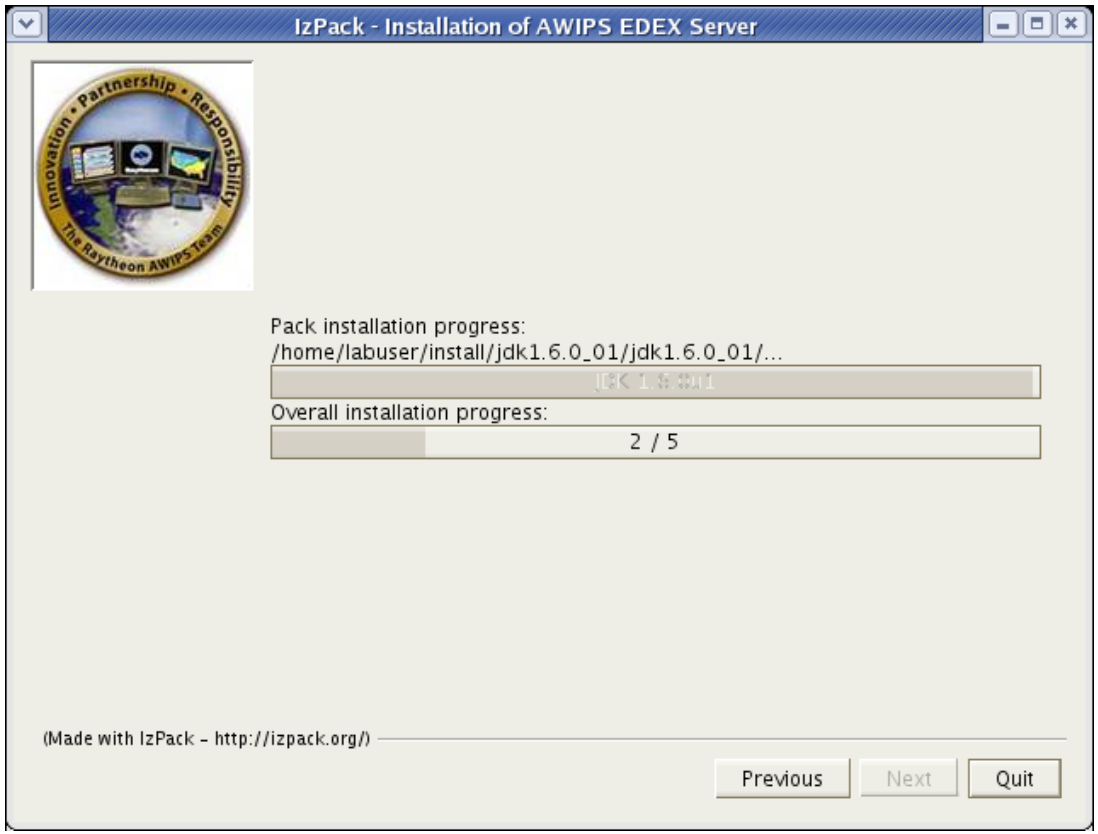
EDEX Installation (Linux) (cont'd)

– Select the shortcuts to install and select Next



EDEX Installation (Linux) (cont'd)

- Installation will begin, and its progress will be displayed
  - When complete, select the Next button

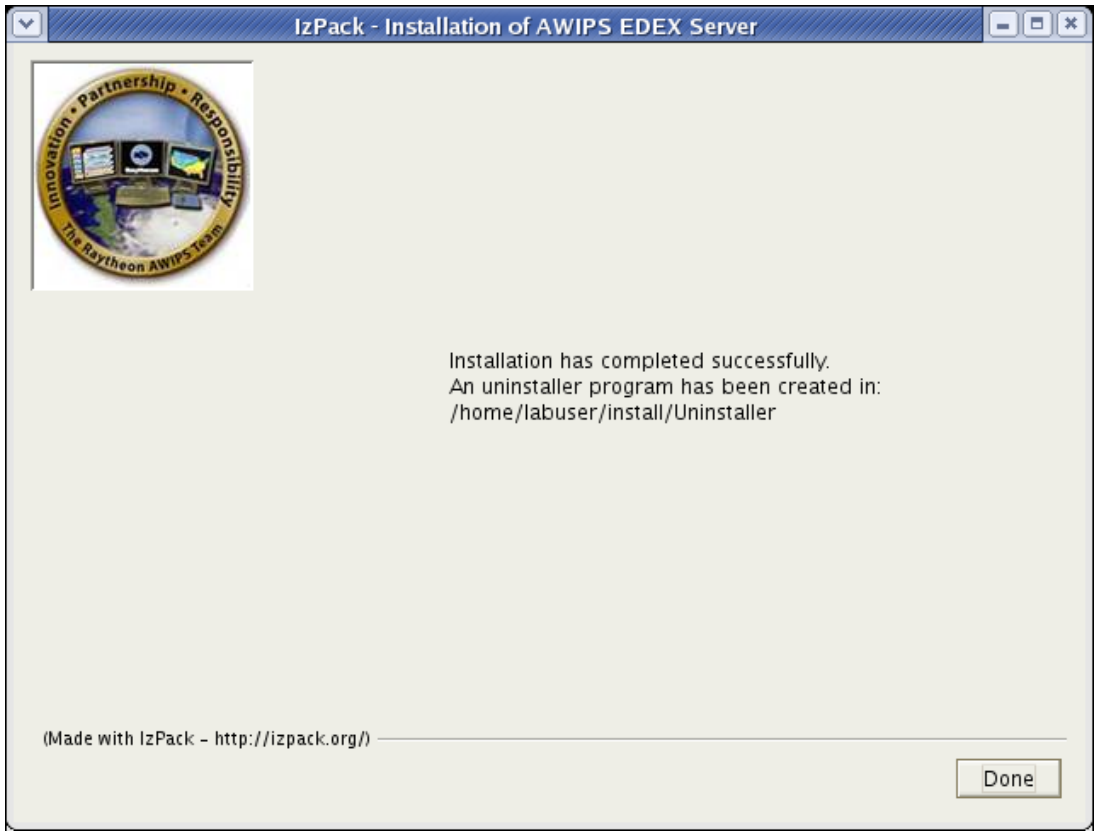


EDEX Installation (Linux) (cont'd)

- PostgreSQL installation will begin
  - When complete, select the Next button

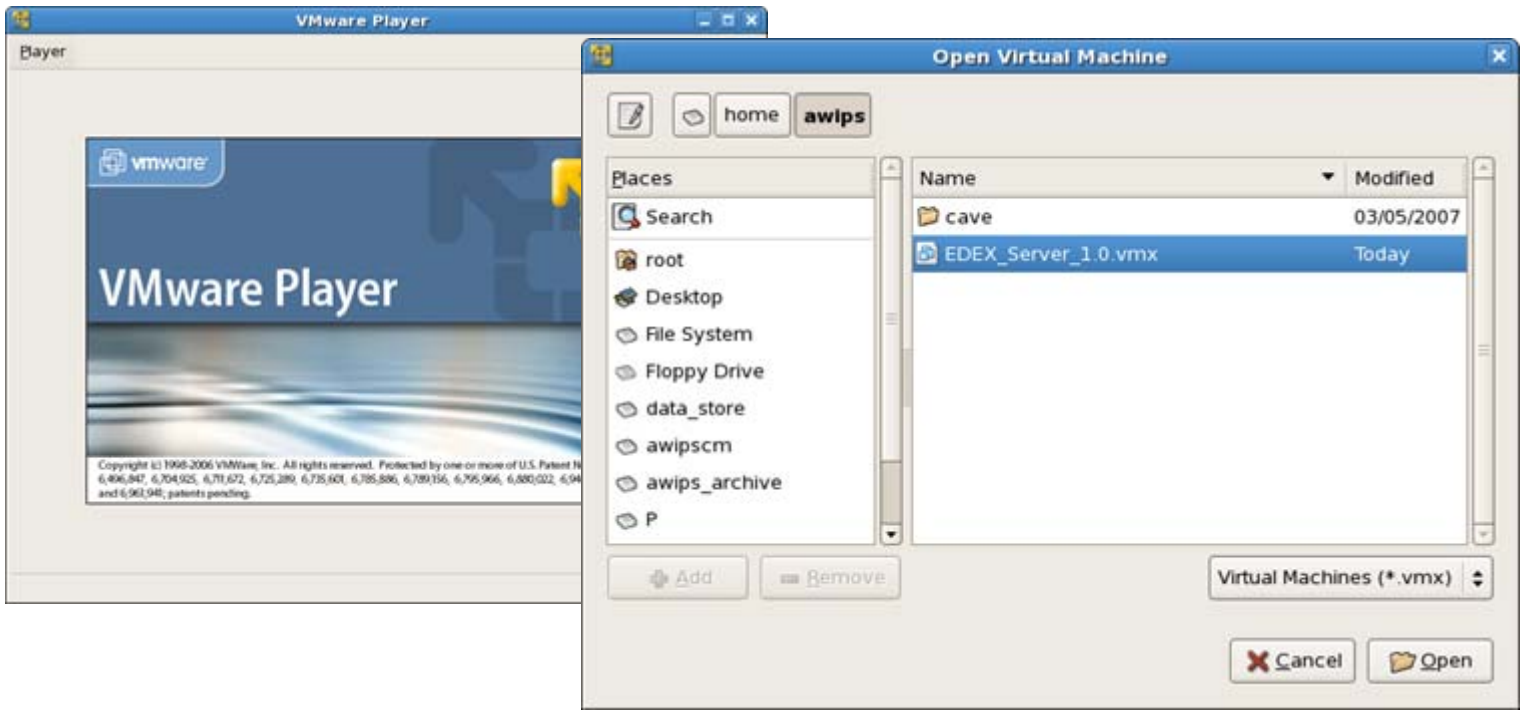
EDEX Installation (Linux) (cont'd)

- Installation is complete
  - Select Done to exit the installer



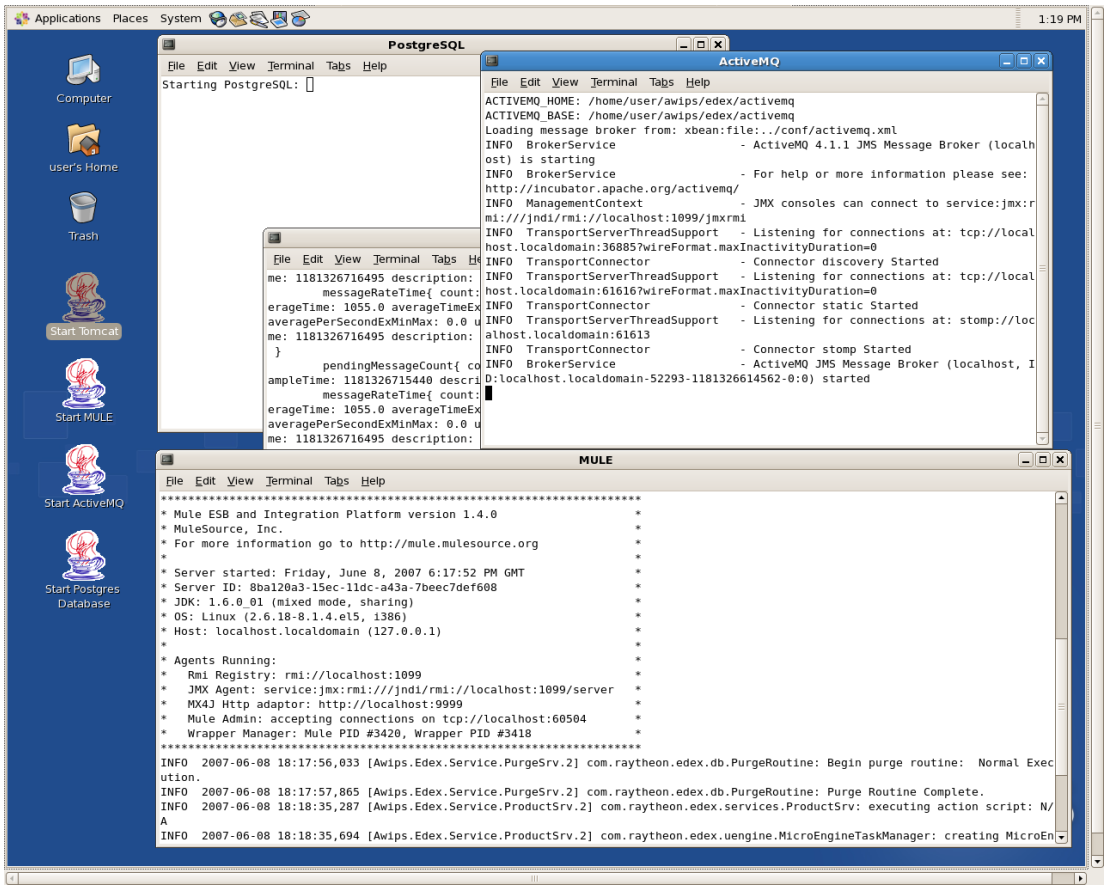
EDEX Installation (VMWare Player)

- EDEX VMWare image requires use of VMWare Player (1.0.3)
  - Locate the VMWare image on the distribution
  - Once located, start the VMWare player and open the file



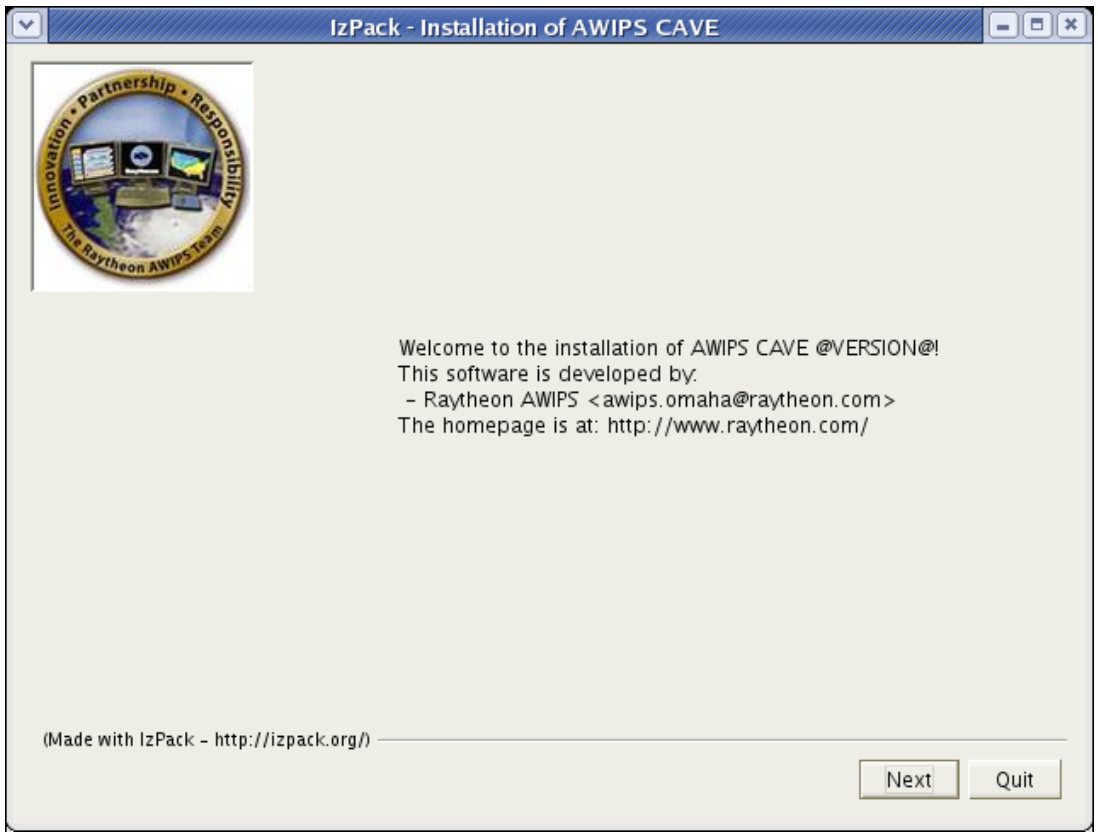
EDEX Installation (VMWare Player) (cont'd)

- VMWare Player will run the image
- EDEX Server available for use from the CAVE application



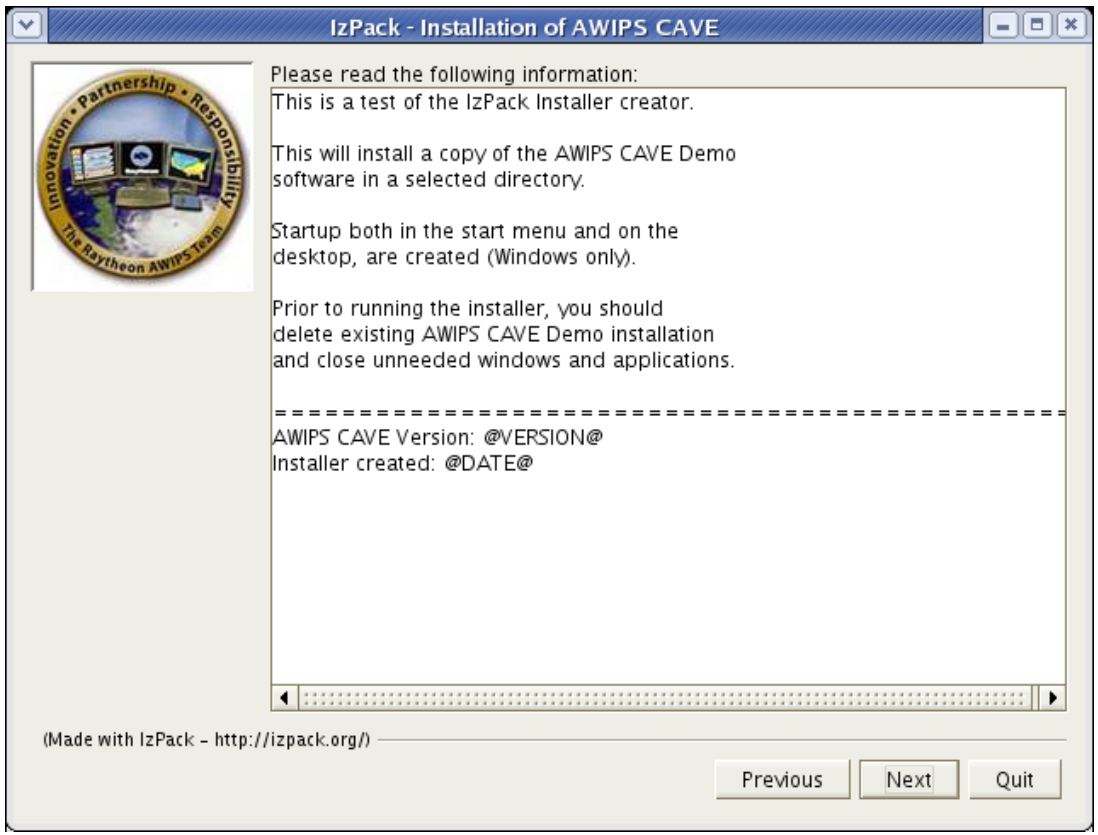
CAVE Installation

- CAVE installer available for use on RHEL 4.2 and Windows XP
  - On the start screen, select Next to proceed



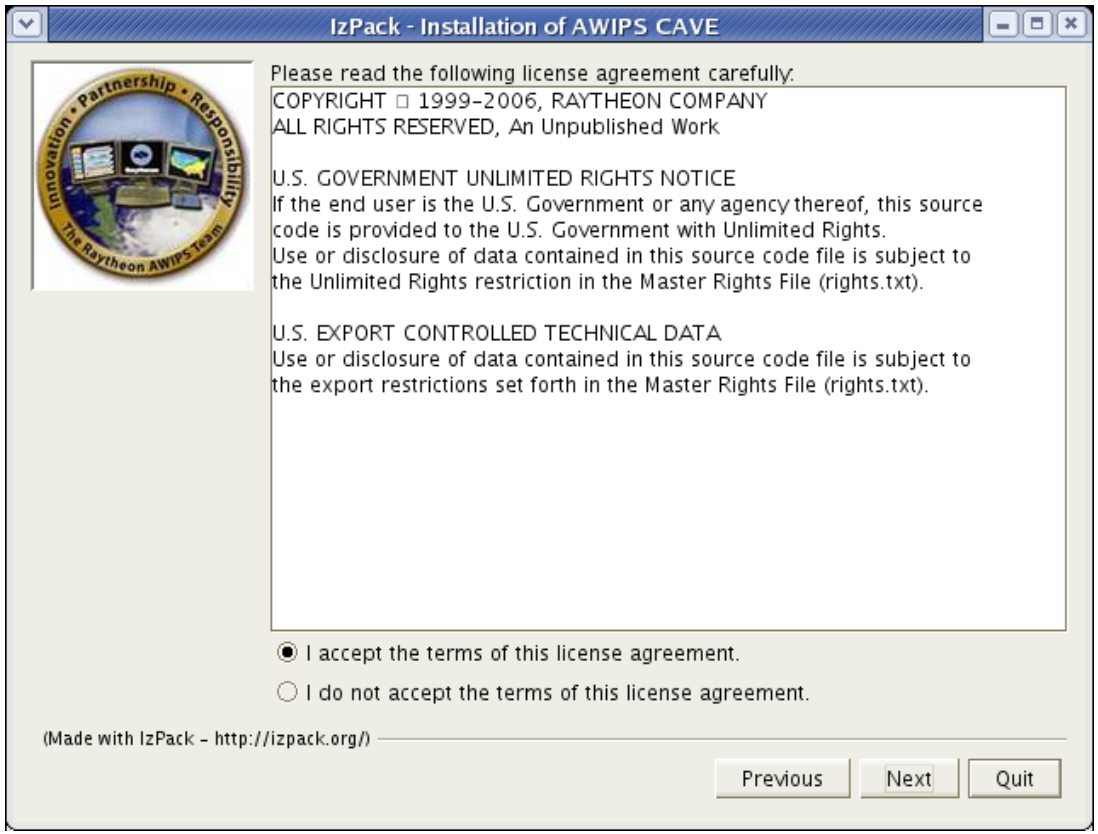
CAVE Installation (cont'd0

– Select Next on the information screen



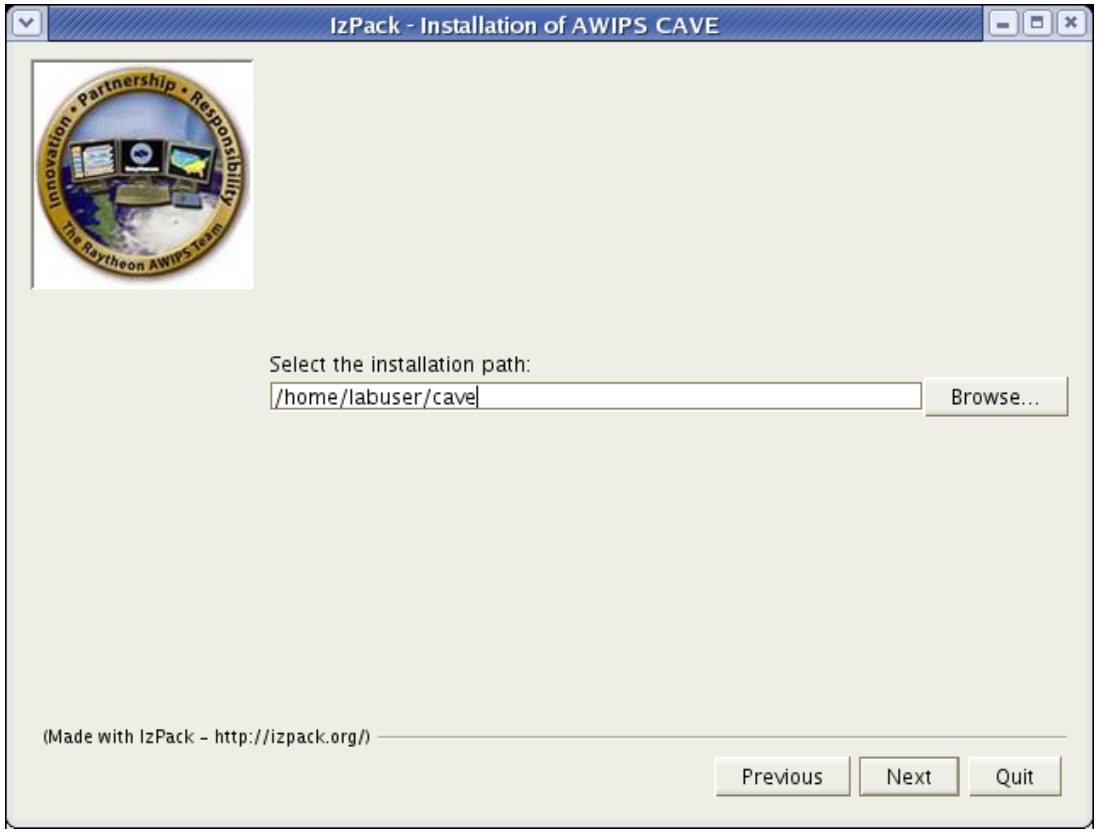
CAVE Installation (cont'd)

– Accept the license agreement and select Next



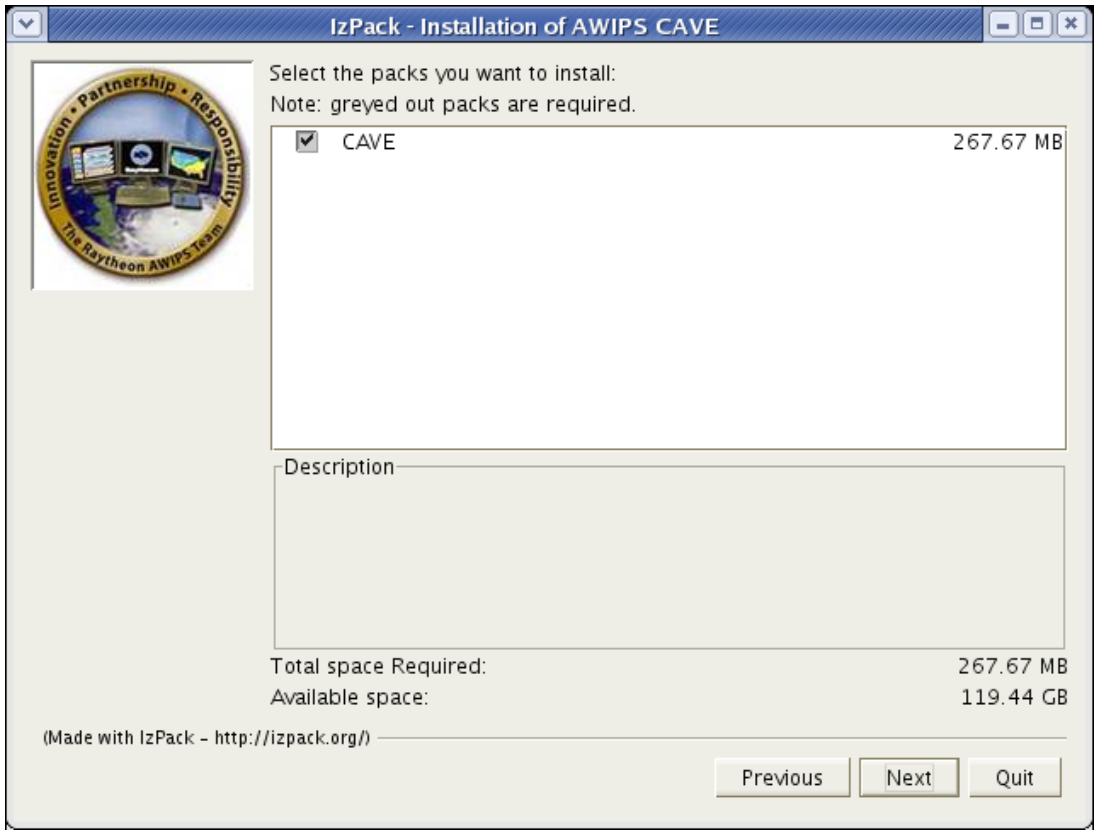
CAVE Installation (cont'd)

– Enter the installation path (or browse to it) and select Next



CAVE Installation (cont'd)

- CAVE is a required component
  - Select Next to continue

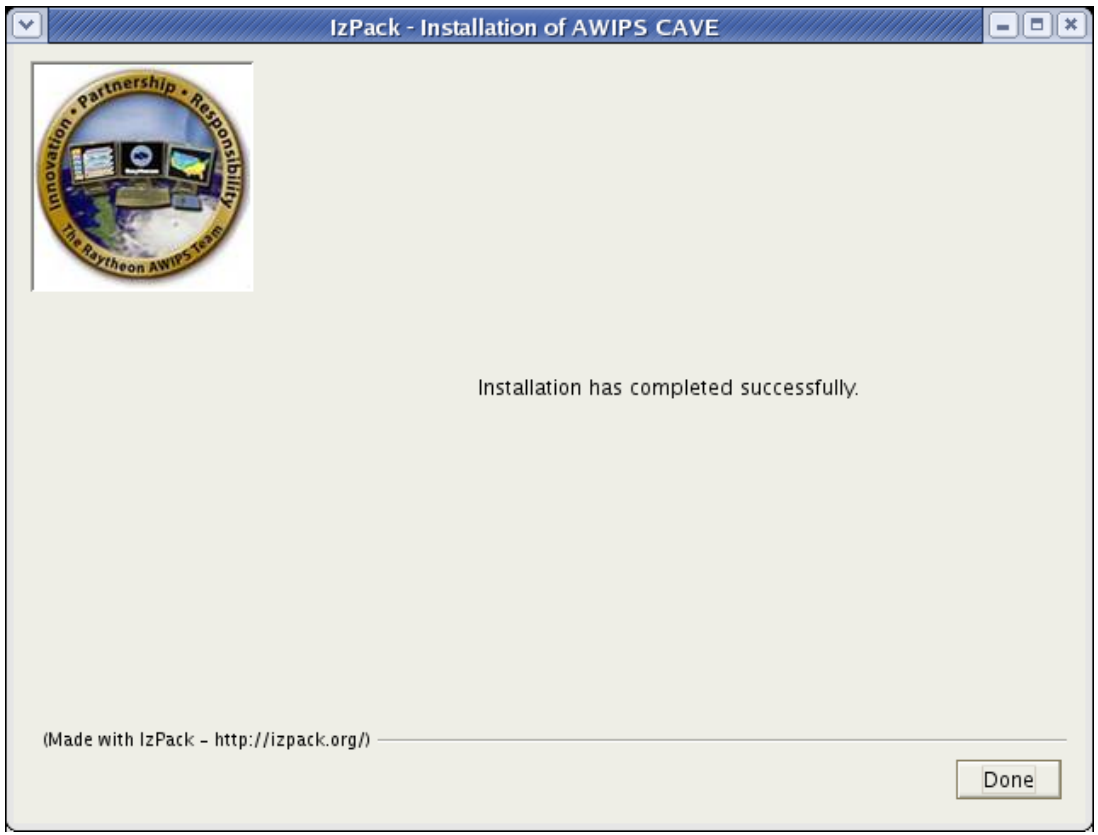


CAVE Installation (cont'd)

- Specify the address of the EDEX server and the location of the HDF5 files
- Select Next to continue

CAVE Installation (cont'd)

- Installation is complete
  - Select Done to exit the installer



Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 10: CAVE Menu Creation (rev. 1)

February 14, 2008

AWP.TRG.SWCTR/TO6.ADE/CAVE10.01

Prerequisites/Objectives

- Prerequisites
  - Familiarity with CAVE baseline (TO5 Briefing)
  - Familiarity with Java and Eclipse
  - Exploration of the CAVE source code baseline
  - ADE 1.0 installed
- Objectives
  - Describe the changes in menu architecture in TO6
  - Provide an example of creating a new menu in CAVE

Estimated Time: 1 hour

Introduction to CAVE Menus

CAVE Menus

- Previous versions of CAVE defined menus in Java using Eclipse 3.2
  - Not flexible enough, not dynamic enough
- Eclipse 3.3: Far more flexible and dynamic menu capability
  - Although currently in a beta state, the benefits of 3.3 architecture are dramatic. Therefore, we chose to develop TO6 baseline against a pre-release version of 3.3
  - New approach has flexible ways of defining and placing menus, dynamically decorating menus, and controlling visibility aspects
  - ADE 1.0 version of CAVE built on the Eclipse 3.3m7 release

CAVE Menus (cont'd)

- In CAVE, we primarily leverage the existing Eclipse 3.3 menu capability
  - Powerful capability; not many extensions needed
- Example of menu extension:

```
<extension
      point="org.eclipse.ui.menus">
   <menuContribution
         locationURI="menu:file?after=afterNewGroup">
      <command
            commandId="com.raytheon.viz.shapefile.shapefileImport"
            label="Import Shapefile..."
            style="push">
      </command>
   </menuContribution>
</extension>
```

CAVE Menus (cont'd)

- Important points
  - **dataURI.** Provides capability for more precise placement of menus. Optional parameters "before" and "after" allow menu placement relative to other menus, menu items, and arbitrary group markers.
  - **Commands.** Menu items tied to commands rather than concrete code, which allows for reuse of menu items in different contexts. Also decouples execution logic from menu presentation.
  - **Command Parameters.** Optional parameters to commands allow menu items to pass unique parameters to the command handler logic. Allows a single menu handler to service multiple menu items with different results.

CAVE Menus (cont'd)

- In addition to menus, a developer must define "commands" and "handlers"
  - **Commands.** Abstract concept of an action (no implementation)
  - **Handlers.** Implementation of an action
- Example:

```
<extension
      point="org.eclipse.ui.commands">
   <command
         id="com.raytheon.viz.shapefile.shapefileImport"
         name="Shapefile Import">
   </command>
</extension>

<extension
      point="org.eclipse.ui.handlers">
   <handler
         class="com.raytheon.viz.shapefile.action.ShapefileImportAction"
         commandId="com.raytheon.viz.shapefile.shapefileImport">
   </handler>
</extension>
```

CAVE Menus (cont'd)

- Handler interface example

```
public class ShapefileImportAction extends AbstractHandler {


    @Override
    public Object execute(ExecutionEvent arg0) throws ExecutionException {
          ShapefileWizard wizard = new ShapefileWizard();
          WizardDialog dialog = new WizardDialog(VizApp.getCurrentEditor()
                    .getSite().getShell(), wizard);
          dialog.create();
          dialog.open();
          return null;
    }

}
```

Exercise: Creating a New Menu Item

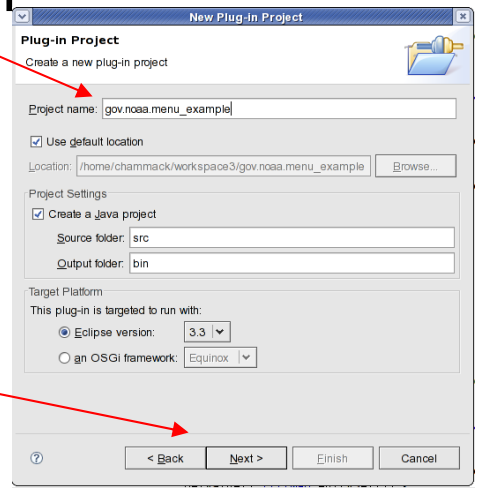
Exercise:
Creating a New Menu Item

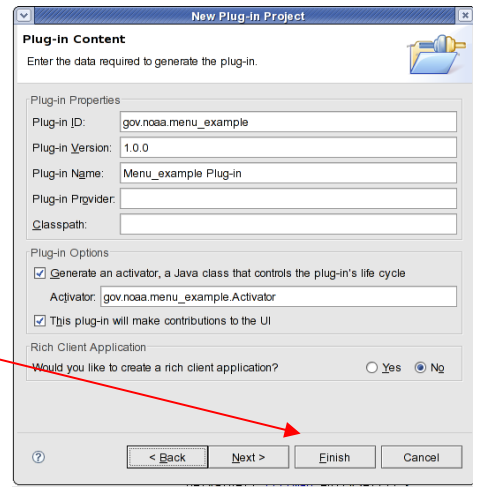
Raytheon

- In this example, we define a new CAVE Plug-In that defines a menu in the new structure. We will:
  - First: Create an action class that defines a specific action
  - Second: Create the command and handler XML to tie into the action
  - Third: Create the menu itself in XML

Exercise: Creating a New Menu Item Getting Started

- Create Plug-In “gov.noaa.menu_example”
- Click “Next”
- Click “Finish”

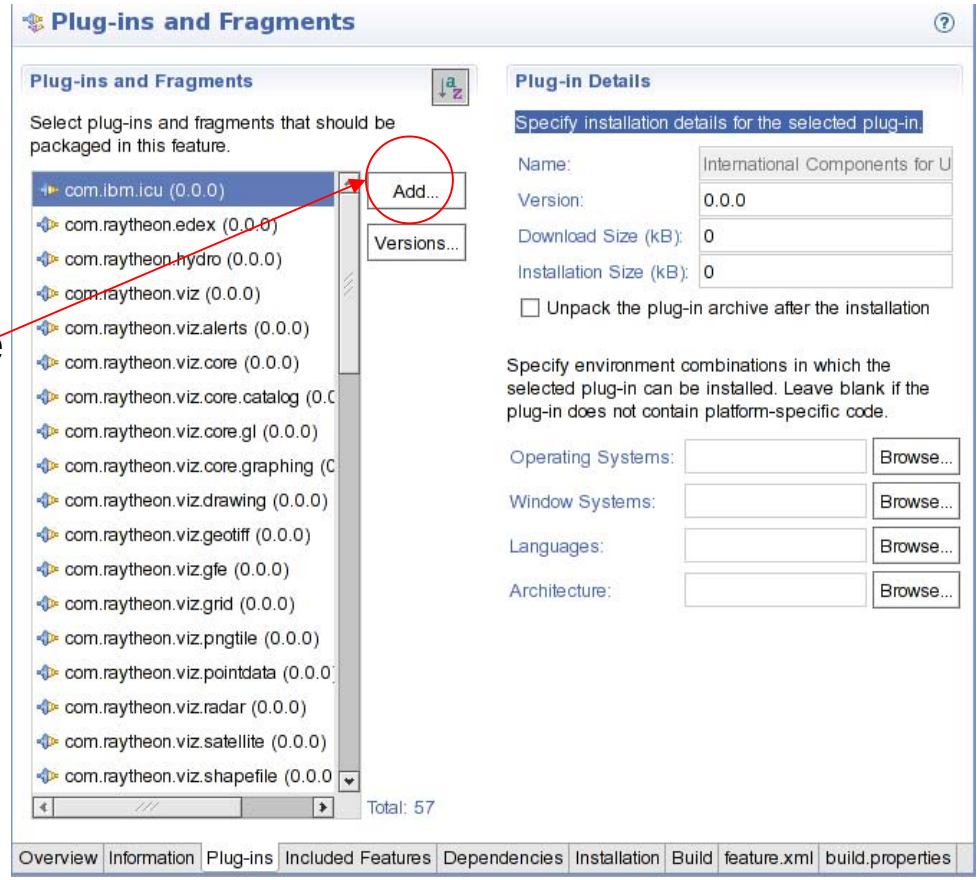




Exercise: Creating a New Menu Item Getting Started (cont'd)

Raytheon

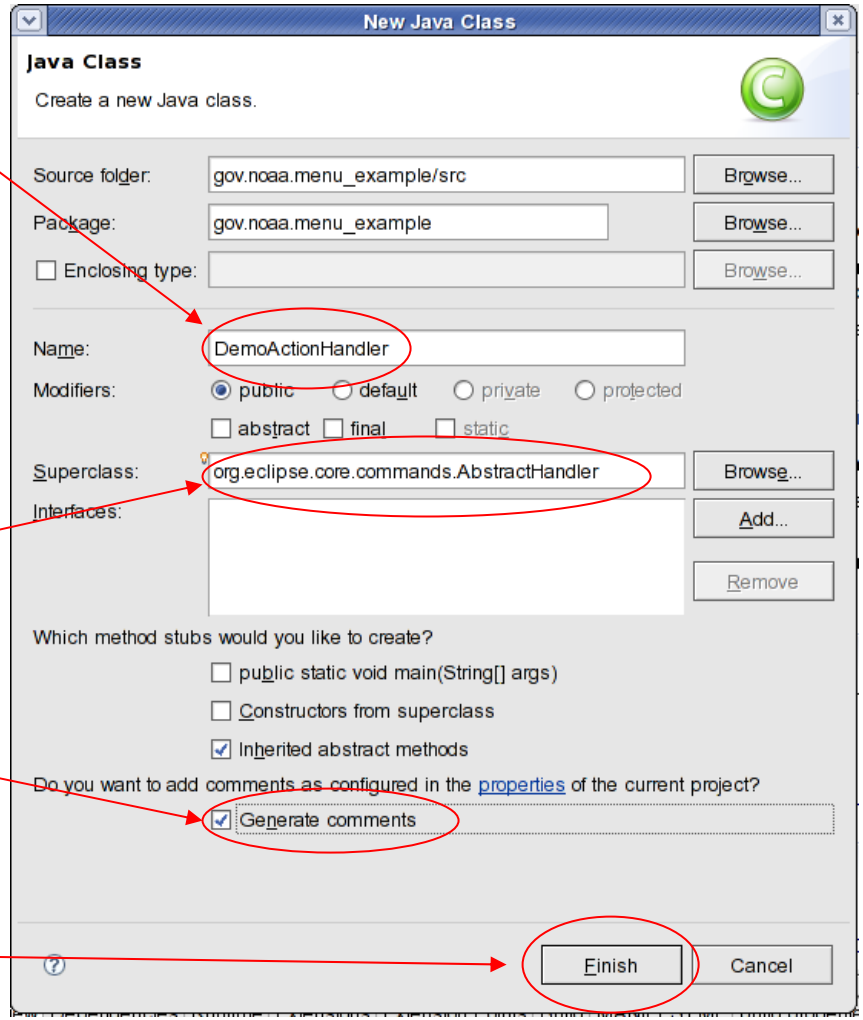
- To allow the new plug-in to participate in the plug-in environment, add the newly created plug-in to the `feature.xml`
- In `com.raytheon.viz.awips`, open `feature.xml`, and choose the Plug-Ins Tab
- Click "Add.."
- Add the newly created Plug-In, and save the configuration



Exercise: Creating a New Menu Item
Creating the Action Class

Raytheon

- Create DemoActionHandler
- Extend AbstractHandler
- Generate comments
- Finish



Exercise: Creating a New Menu Item Creating the Action Class (cont'd)

Raytheon

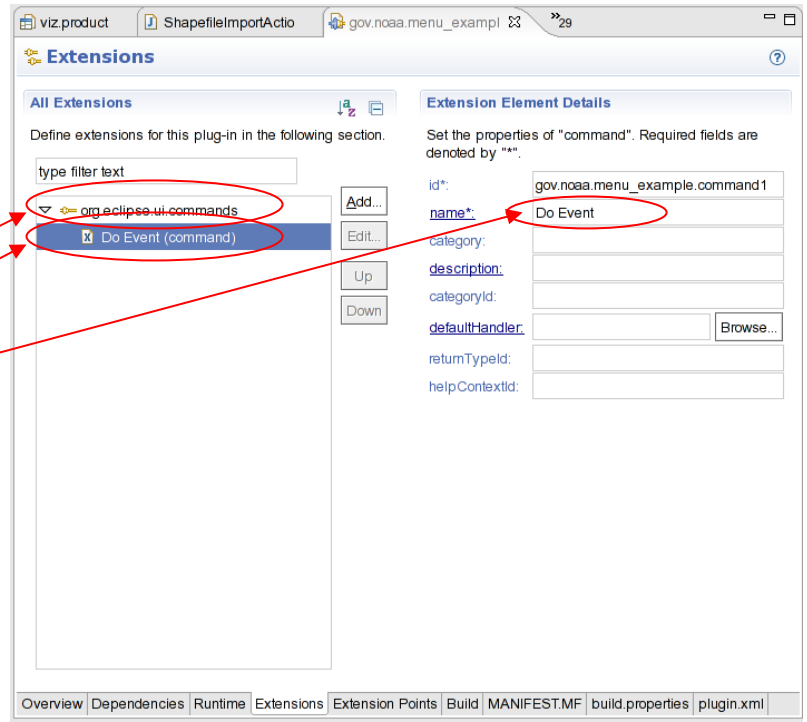
```
public class DemoActionHandler extends AbstractHandler {

   /* (non-Javadoc)
    * @see
   org.eclipse.core.commands.AbstractHandler#execute(org.eclipse.core.comma
   nds.ExecutionEvent)
    */
   @Override
   public Object execute(ExecutionEvent arg0) throws ExecutionException {
      Shell shell = Display.getCurrent().getActiveShell();

       MessageDialog.openInformation(shell, "Hello", "This was triggered by
   "
          + arg0.toString());
      return null;
   }

}
```

Exercise: Creating a New Menu Item Building the Command XML

Raytheon

Using the Eclipse plug-in manifest editor:

- Create org.eclipse.ui.command
- Create “Do Event”



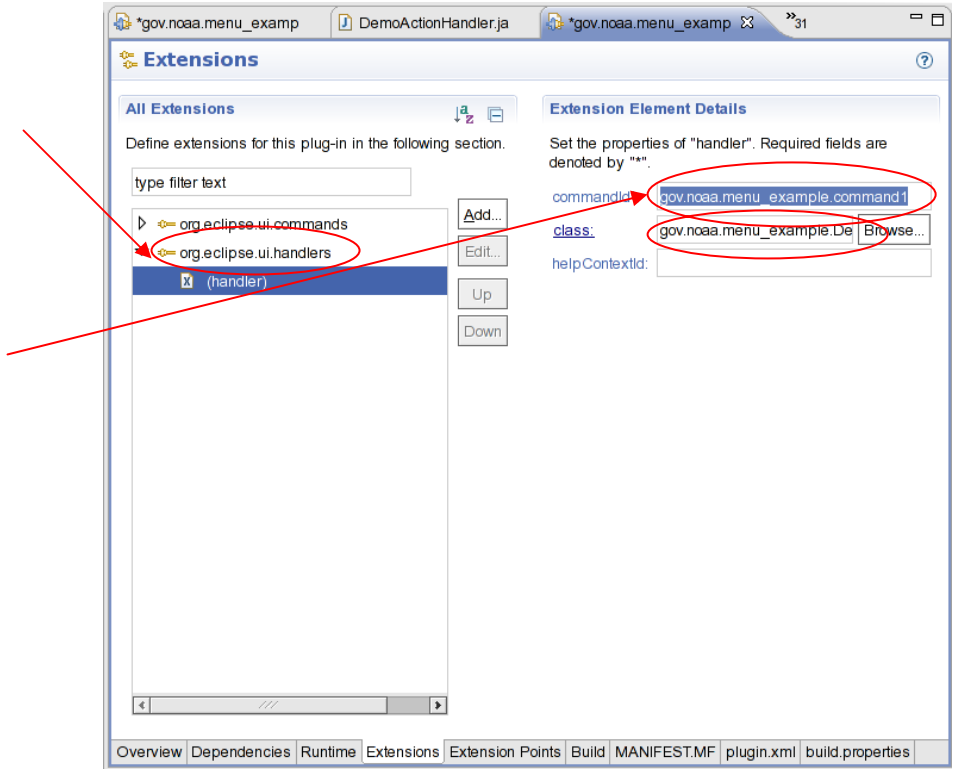
Builds the XML:

```
<extension
        point="org.eclipse.ui.commands">
     <command
           id="gov.noaa.menu_example.command1"
           name="Do Event">
     </command>
 </extension>
```

Exercise: Creating a New Menu Item
Creating the Handler XML

Raytheon

- Create org.eclipse.ui.handlers
- Set commandId to match command created previously
- Point class to DemoActionHandler class

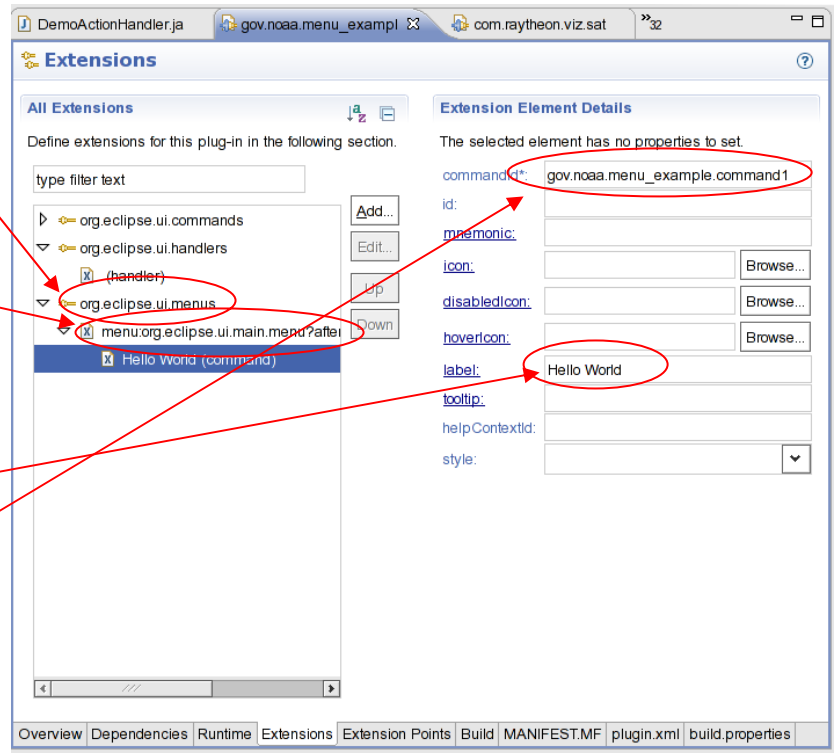


Builds the XML:

```
<extension
        point="org.eclipse.ui.handlers">
     <handler
           class="gov.noaa.menu_example.DemoActionHandler"
           commandId="gov.noaa.menu_example.command1">
     </handler>
  </extension>
```

Exercise: Creating a New Menu Item Creating the Handler XML (cont'd)

- Create org.eclipse.ui.menus
- Create menuContribution:
  menu:org.eclipse.ui.main.menu?after=File
- Create command
  - Set Label "Hello World"
  - Match commandId to previously created command



Builds the XML:

```
<extension point="org.eclipse.ui.menus">
   <menuContribution
         locationURI="menu:org.eclipse.ui.main.menu?after=file">
      <command commandId="gov.noaa.menu_example.command1"
            label="Hello World">
      </command>
   </menuContribution>
</extension>
```

Wrap-Up

Summary

- Eclipse 3.3 provides support for menu placement and dynamic menus that is far superior to previous Eclipse incarnations
- Best reference: The (forthcoming!) Eclipse 3.3 documentation

Resources

- On the ADE 1.0 DVD
  - Current code available for examination in the CAVE baseline
  - JavaDoc documentation available

Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 11: Localization (rev. 1)

February 14, 2008

AWP.TRG.SWCTR/TO6.ADE/CAVE11.01

Prerequisites/Objectives

- Prerequisites
  - Familiarity with CAVE baseline
  - Familiarity with Java and Eclipse
  - Exploration of the CAVE source code baseline
  - ADE 1.0 installed
- Objectives
  - Introduce the localization concepts in ADE 1.0
  - Describe the new localization process

Estimated Time: 1 hour

General Localization Approach

- Localization procedure should occur at startup
  - CAVE localization is simple: Go into preferences, choose a new localization, and restart CAVE
  - EDEX localization is simple: Choose a configuration during the installation process

    *[**Note:** In the future, a technique to change server localization after installation may be provided (likely of very limited value).]*
- Requires a different approach to data:
  - Less subsetting required because CAVE can work with much larger datasets
    - Example: No longer necessary to create a unique localized version of a state's shapefile
  - Every site should work from the master set of data when possible. Subsetting, if required, should occur on the first data access (in the regular processing procedure)

Localization Overview

- Three major localization components in ADE 1.0
  - Localization service (present on the EDEX server)
    - Stores localization preferences for CAVE
  - CAVE Localization Preferences
    - Contains all of the localization preferences for the workstation
  - EDEX Localization Preferences
    - Stores primarily site-specific configuration options
    - Mostly related to system configuration, so “relocalizing” – although useful from a testing perspective – is likely to be of limited use in a deployment sense

Localization Overview

Base:

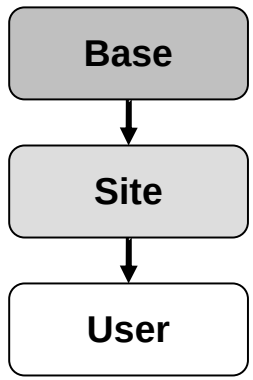
```
<configuration>
   <textureCardPreference>128</textureCardPreference>
   <textureMemoryPreference>384</textureMemoryPreference>
   <framesPerSecondPreference>25</framesPerSecondPreference>
   <tileBoundaries>false</tileBoundaries>
   <connectionMethod>jms</connectionMethod>
   <jmsServerAddress>tcp://localhost:61616</jmsServerAddress>
   <dataDirectory>/awips/opt/data/hdf5</dataDirectory>
   <fontMagnification>1.0</fontMagnification>
</configuration>
```

Site:

```
<configuration>
   <siteName>KOAX</siteName>
   <siteFullName>Omaha</siteFullName>
   <siteType>WFO</siteType>
   <dataDirectory>/oax-awips/opt/data/hdf5</dataDirectory>
</configuration>
```

User:

```
<configuration>
   <fontMagnification>1.25</fontMagnification>
</configuration>
```

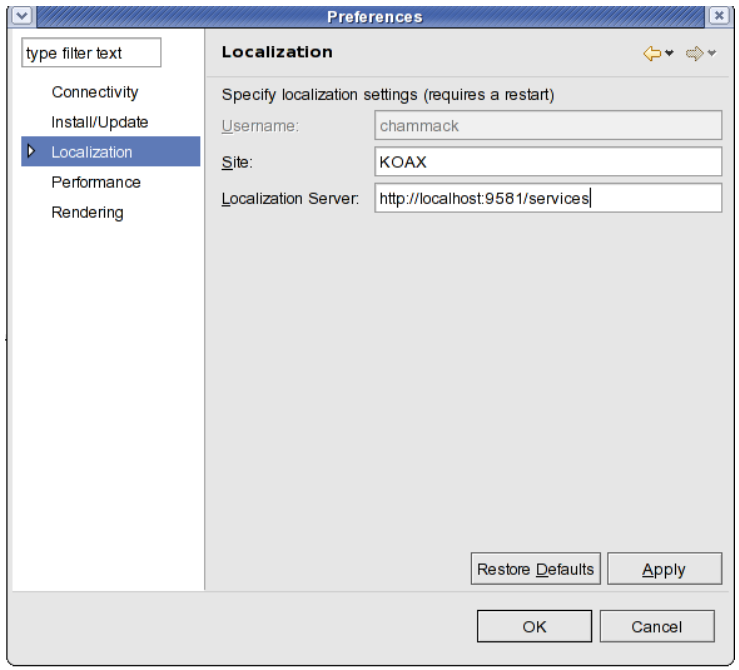


CAVE Localization Preferences

- Eclipse already has a built-in concept of preferences. So why change?
  - Default Eclipse preferences have no concept of base, site, and user localization – just a single level
  - There is no concept of preference synchronization – changing an option on one workstation doesn't will not change it on other workstations
  - Finally . . . We are continuing to use Eclipse preferences; we are just extending their existing capabilities
    - Instead of using the Eclipse preference store, preferences are now stored in an easily accessible XML format

Localization in CAVE

- Basic Concept
  - Your currently logged-in Linux (or Windows) username determines your user context
    - However, this could become a manual selection process if necessary
  - The site choice could be predefined for an installation, but easily changed in Preferences



Localization in CAVE

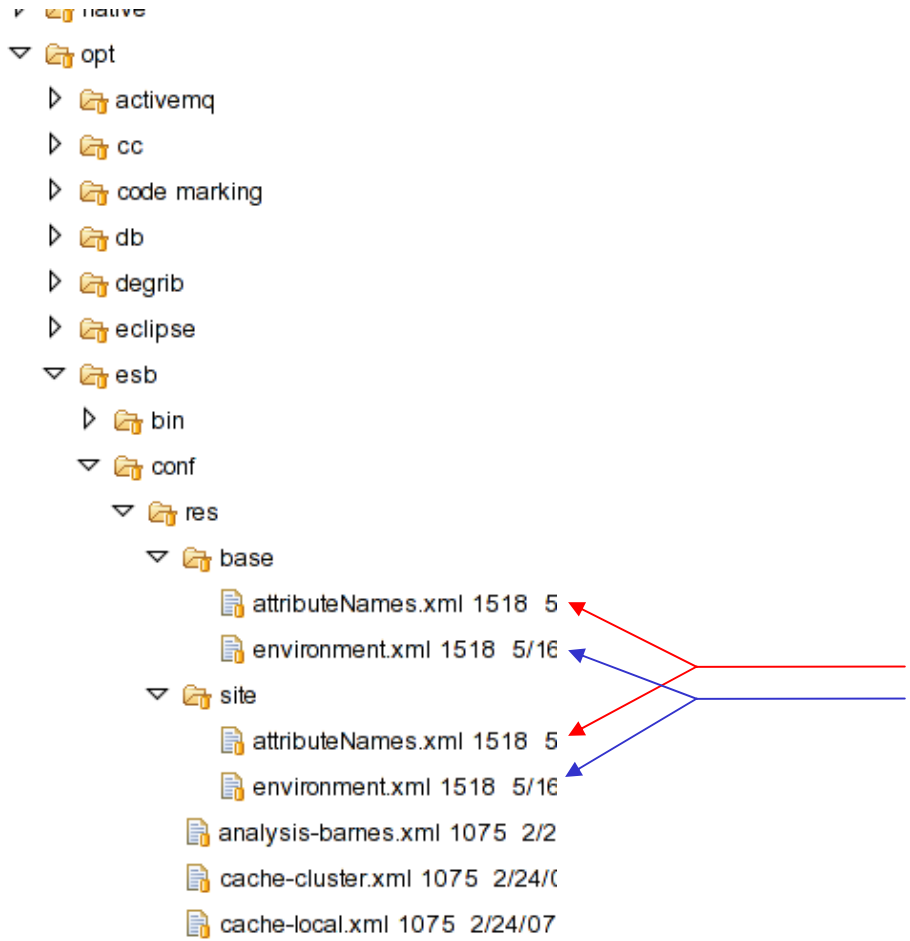
- At start-up, CAVE will contact the localization service
- Some potential synchronization items
  - Preferences
    - Example: Site name, local grid windows
  - Menus
    - Example: Site-specific data menus
  - Data
    - Example: Colormaps, parameter mappings, etc.
  - Future
    - Custom datatype plug-ins (custom code!)

Localization in EDEX

- Because localization primarily occurs at the workstation (to facilitate the possibility of servicing multiple locales with a single server configuration), EDEX localization is primarily system configuration-based
- Configuration files split into two parts
  - **Base:** Contains the stock configuration values
  - **Site:** Contains any values that the site chooses to override for its locale

Note: *Site files may be empty, indicating that the default system configuration should be used.*

Localization in EDEX



Note that same filename appears in “base” and “site” contexts

Summary

- Localization provided through two simple, unified interfaces
  - Configuration for the server
  - Localization for the client, with server synchronization capability
- Localization provides a multi-tiered configuration
  - Base, Site for Server
  - Base, Site, and User for Client

Resources

- On the ADE 1.0 DVD
  - Current code available for examination in the CAVE base-line
  - JavaDoc documentation available

Raytheon

Customer Success Is Our Mission

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 12: TO8 ADE 1.0 Developer Updates

February 14, 2008

AWP.TRG.SWCTR/TO8.ADE/CAVE12.00

Objectives

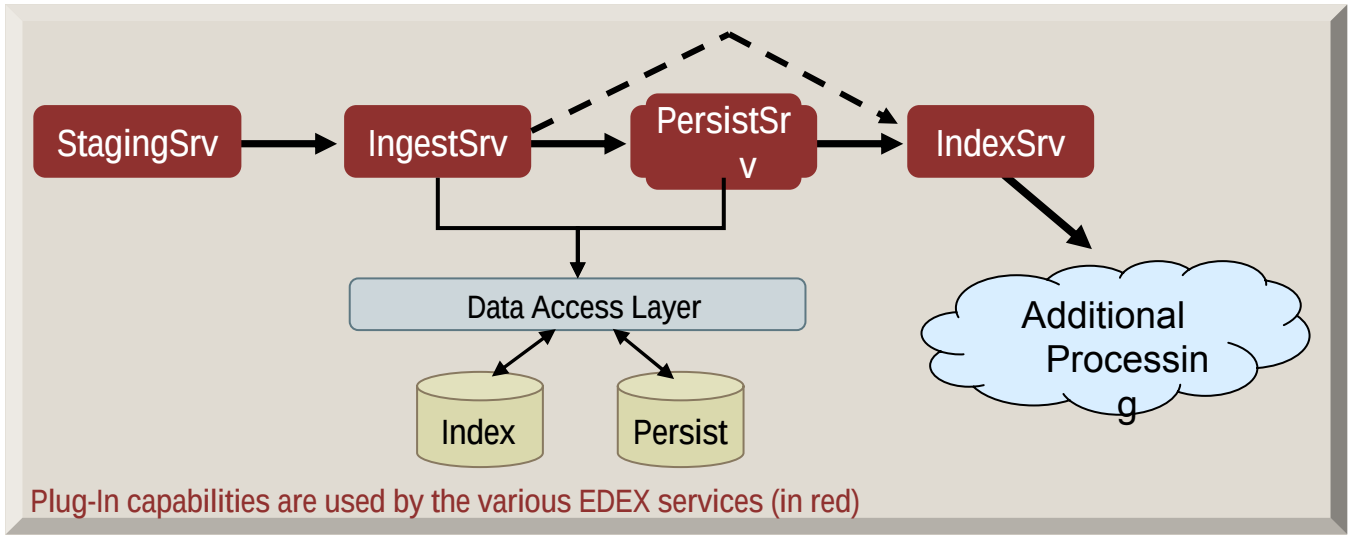
- Provide updates to previously covered developer topics including
  - Modification to EDEX Ingest data flow and impact to data-type plug-in development
  - Modification to EDEX Data Access Layer and impact to data-type plug-in development
  - Modification to EDEX Data Base Definition pattern and impact to data-type plug-in development
  - Modifications to the EDEX Plug-in Creation Tool
  - Additions to the AWIPS II installers
  - Preview of upcoming training and capabilities

Revised Ingest Data Flow

Modified EDEX Ingest Data Flow

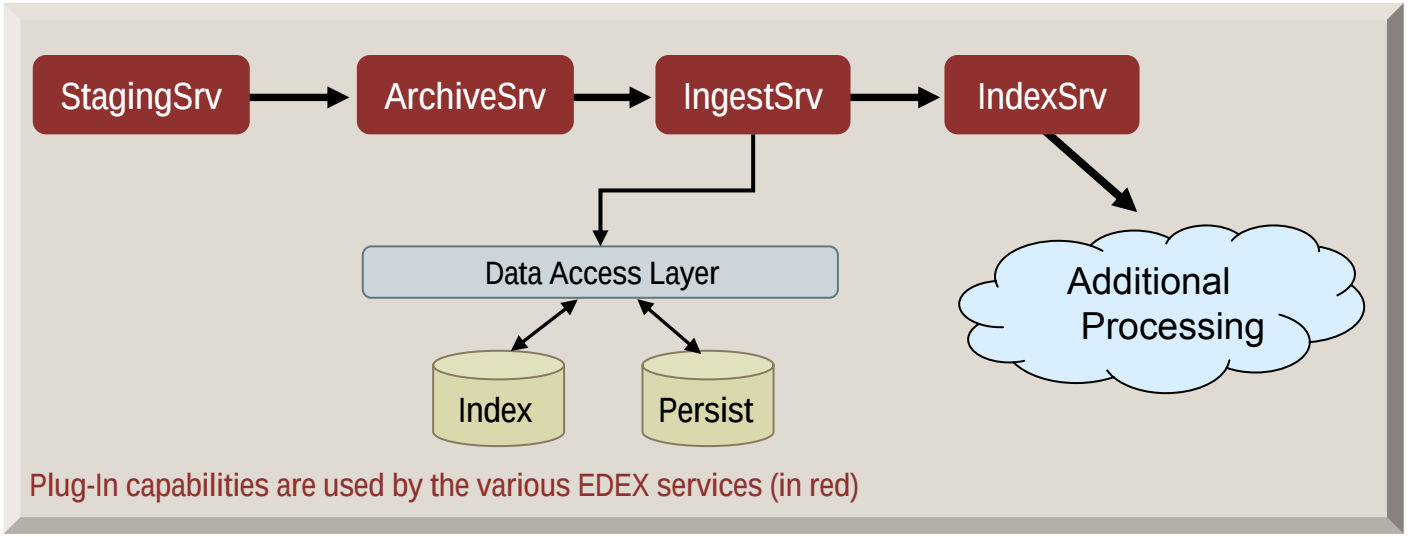
- Existing Ingest Data Flow was causing memory-related problems
  - This is critical to system stability in the future.
- For TO8, EDEX Ingest DATA Flow is simplified and standardized
- Standardized: All data initially handled by a staging service to facilitate automated load balancing in a clustered environment
- Standardized: All data passes through an archive service to allow archiving of data for future playback
- Simplified: Ingest and persistence services combined into a single service
- Simplified: Standardization of data flows simplifies the configuration of data endpoints

Ingest Data Flow Delivered in TO 6



- Features separate mule endpoints for:
  - Reading data files (StagingSrv)
  - Decoding the data (IngestSrv)
  - Saving Data to HDF-5 (PersistSrv)
  - Writing meta-data to the database (IndexSrv)
- Not all data processed by PersistSrv

Ingest Data Flow Delivered in TO 8



- New service (ArchiveSrv) added to provide archiving of inbound data
- IngestSrv now handles both decoding of data and persistence to HDF-5 based data store
- StagingSrv, ArchiveSrv and IngestSrv have a separate end-point for each data type

TO 8 EDEX Ingest Services

Service	Description
StagingSrv	Listens on an SOA endpoint for data and moves the data to working directory on a network shared device. Initially places file path on JMS queue to allow cluster-based processing.
ArchiveSvr	Listens to JMS queue for available work. Copies files from working directory to archive directory. Returns file path to JMS queue for further processing.
IngestSrv	Listens to JMS queue for available work. Reads file from working directory and decodes file contents. Persists certain binary data to HDF-5 archive. Passes metadata to JMS queue for further processing.
IndexSrv	Listens to JMS queue for available work. Saves the metadata extracted from the ingested data to PostgreSQL database for client retrievals. Passes meta-data to JMS queue for further processing.

Data Flow Impact

- First three stops of the modified data flow configured in a single XML file in each data-type plug-in
  - XML file is the xxx-ingest.xml file, located in the "res/endpoints" directory of the plug-in (xxx represents the data type, e.g. satellite)

    ***[Note**: See next few slides for the default configuration for the satellite plug-in.]*
- Final stop of the data flow configured in the Index Server's configuration file, index.xml
  - This file is located in "opt/esb/conf"
- All configuration files located in the EDEX baseline in the ADE

Demo: Use Eclipse file browser to show location.

satelite-ingest.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mule-configuration PUBLIC "-//SymphonySoft //DTD mule-configuration XML V1.0//EN"
   "http://www.symphonysoft.com/dtds/mule/mule-spring-configuration.dtd">

<mule-configuration version="1.0">
   <model name="edex" type="seda">

   <!-- Endpoint to stage Satellite data -->

   <!-- Endpoint to archive Satellite data -->

   <!-- Endpoint to ingest Satellite data -->

   </model>
</mule-configuration>
```

Add configuration for endpoints.

This is the basic SOA configuration file. Content represented by the three comments is shown on the next three slides.

satelite-ingest.xml (cont'd)

```
<!-- Endpoint to stage Satellite data -->
<mule-descriptor
     name="Awips.Edex.Service.StagingSrv-satellite"
     singleton="true"
     implementation="com.raytheon.edex.services.StagingSrv
     outboundEndpoint="jms://ar/sat">
   <inbound-router>
      <endpoint name="sbnSatIngestEndpoint"
                address="file://../../data/sbn/sat/?transformers=NoActionTransformer">
         <properties>
            <property name="moveToDirectory"
                      value="../../processing" />
         </properties>
      </endpoint>
   </inbound-router>
   <threading-profile maxThreadsActive="1" maxThreadsIdle="1" />
</mule-descriptor>
```

EDEX Class handling message.

Outbound JMS queue.

Inbound source directory.

Move to working directory.

Single instance used for this end point.

This is configuration for the satellite staging service. It is configured to "sniff" a directory and move each file to the working directory.

satelite-ingest.xml (cont'd)

```
<!-- Endpoint to archive Satellite Data -->
<mule-descriptor
     name="Awips.Edex.Service.ArchiveSrv-satellite"
     singleton="false"
     implementation="com.raytheon.edex.services.ArchiveSrv"
     outboundEndpoint="jms://cp/sat">
   <inbound-router>
      <endpoint name="AR-Sat" address="jms://ar/sat" />
   </inbound-router>
   <threading-profile maxThreadsActive="4" maxThreadsIdle="4" />
   <properties>
      <property name="pluginName" value="SATELLITE" />
      <property name="archiveDirectoryLocation"
                value="../../data/archive/sat/" />
      <property name="jmxModeOn" value="true" />
   </properties>
</mule-descriptor>
```

EDEX Class handling message.

Outbound JMS queue.

Inbound JMS queue.

Up to 4 instances used for this end point.

Move to working directory.

This is the configuration for the satellite archive service. It is configured to listen on a JMS queue and process the file identified by the message received from that queue.

satelite-ingest.xml (cont'd)

```
<!-- Endpoint to ingest Satellite data -->
<mule-descriptor
     name="Awips.Edex.Service.IngestSrv-satellite"
     singleton="false"
     implementation="com.raytheon.edex.services.IngestSrv">
   <inbound-router>
      <endpoint name="CP-Sat" address="jms://cp/sat" />
   </inbound-router>
   <outbound-router>
      <router className="org.mule.routing.outbound.FilteringListMessageSplitter">
         <endpoint address="vm://indexVMQueue" />
      </router>
   </outbound-router>
   <threading-profile maxThreadsActive="1" maxThreadsIdle="1" />
   <properties>
      <property name="pluginName" value="SATELLITE" />
   </properties>
</mule-descriptor>
```

EDEX Class handling message.

Inbound JMS queue.

Outbound JMS queue, splits multiple messages.

Thread pooling for this endpoint.

This is the configuration for the satellite ingest endpoint. It is configured to listen on a JMS queue and process the file identified by the message received from that queue.

Questions?

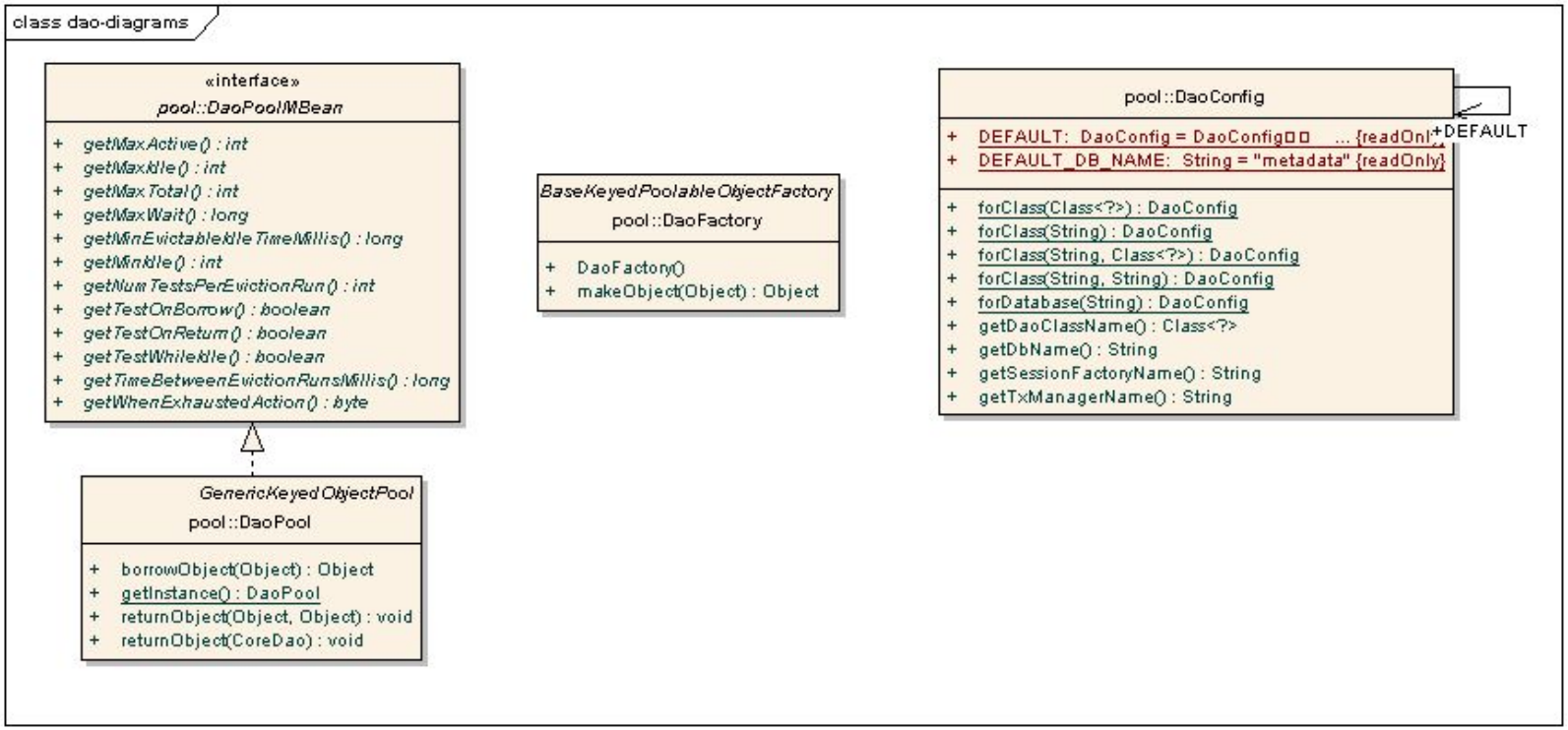

Innovation • Partnership • Responsibility
The Raytheon Missile Team

Revised Data Access Layer Implementation

DAO Pooling: Overview

- Database performance, specifically slow database access times, identified as an issue coming out of TO6
  - The Data Access Layer (DAL) was instantiating and destroying a Data Access Object (DAO) for each database interaction
  - Because the DAO encapsulated the database connection, this tends to be a (time-) expensive operation
- Pooling mechanism introduced in TO8 to limit the potentially expensive operation of instantiating data access objects
  - DAO created the first time it is needed, then maintained in a pool of available objects
  - When a client needs to access the database, it gets the DAO from the pool rather than creating a new DAO object instance

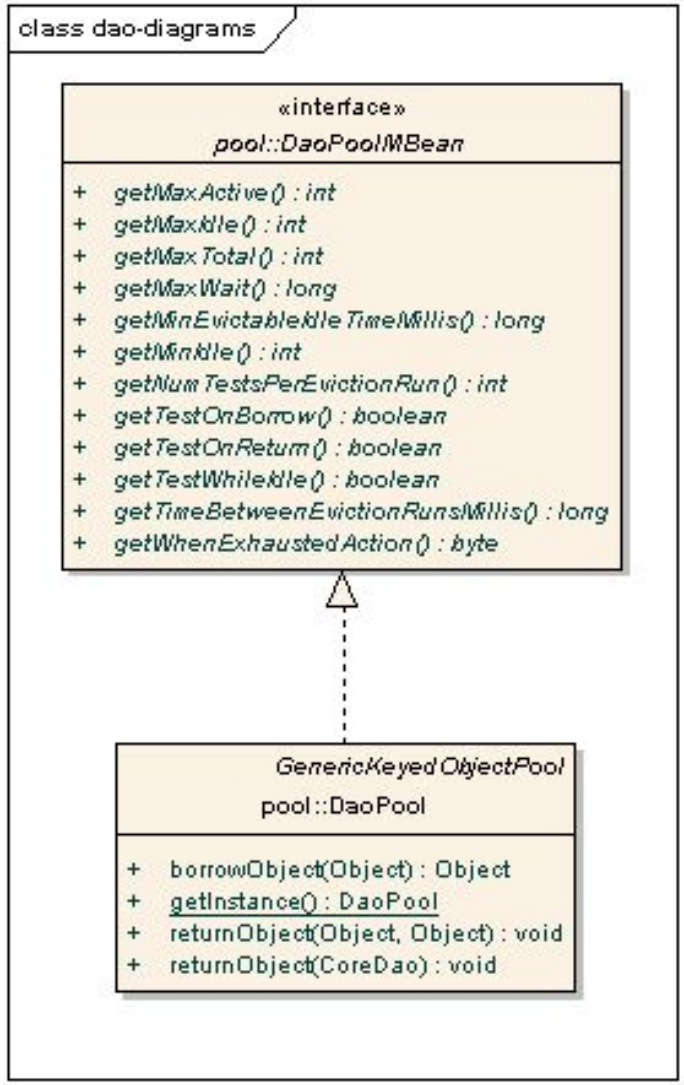
DAO Pooling: Overview (cont'd)



- DAO pooling mechanism consists of three classes
  - DaoPool: A singleton, this is the class clients utilize to borrow and return a DAO
  - DaoFactory: The class is used internally by DaoPool to instantiate appropriate DAO
  - DaoConfig: This class contains information about the DAO; the database name, Hibernate class, and the Hibernate session factory and transaction factory. Provides static convenience methods for obtaining configurations.

DAO Pooling: DaoPool Class

- Main class used for borrowing data access objects
- Consists of two main methods:
  - borrowObject(). Used to obtain the DAO from the pool
  - returnObject(). Used to return the DAO to the pool
- Every call of borrowObject() must be followed by a matching call of returnObject()



DAO Pooling: Borrowing a DAO

- Make a call to DaoPool.getInstance().borrowObject(Object key);
- Key may be any of 3 types object types: String, Class<?> and DaoConfig
  - String object: The value of the String is the name of a database
    - A data access object for the specified database is returned
    - The DAO will be usable only for database inserts and updates (Hibernate requires a class instance to retrieve data)
  - Class<?> object: The Class object of a DAO
    - An instance of the DAO is returned
    - The DAO has full database functionality available
  - DaoConfig object: An instance of DaoConfig describing the desired DAO
    - The DaoFactory is used to construct a DAO matching the properties of the DaoConfig object.
    - This option allows user to create a DOA “on the fly” when the DAO does not already exist

DAO Pooling: Utilization

This code snippet is from the AIREP Decoder.

- As with any pooled resource, one must be careful to return borrowed objects to the pool.

```
AirepDao dao = (AirepDao) DaoPool.getInstance()
               .borrowObject(AirepDao.class);
try {
   // set up dataURI
   Object [] dbRpt = dao.queryByDataURI(dataURI);
   // use the results of the query
} catch (Exception e) {
   // Handle Exception
} finally {
   DaoPool.getInstance().returnObject(dao);
}
```

- Always use a **try – finally** block when using the DAO pool
- As shown here, the DAO class object is normally used to borrow the DAO instance

DAO Pooling: Pool Configuration

- Pool Configuration options located in the DALConfig.xml file
- These may be changed, but will not take effect until Mule is restarted.

```
<property name="props">
   <props>
      <prop key="maxActive">150</prop>
      <prop key="maxIdle">50</prop>
      <prop key="maxTotal">200</prop>
      <prop key="maxWait">-1</prop>
      <prop key="minEvictableIdleTimeMillis">10000</prop>
      <prop key="minIdle">0</prop>
      <prop key="numTestsPerEvictionRun">10</prop>
      <prop key="testOnBorrow">false</prop>
      <prop key="testOnReturn">false</prop>
      <prop key="testWhileIdle">false</prop>
      <prop key="timeBetweenEvictionRunsMillis">60000</prop>
   </props>
</property>
```

DAO Pooling: Additional Information

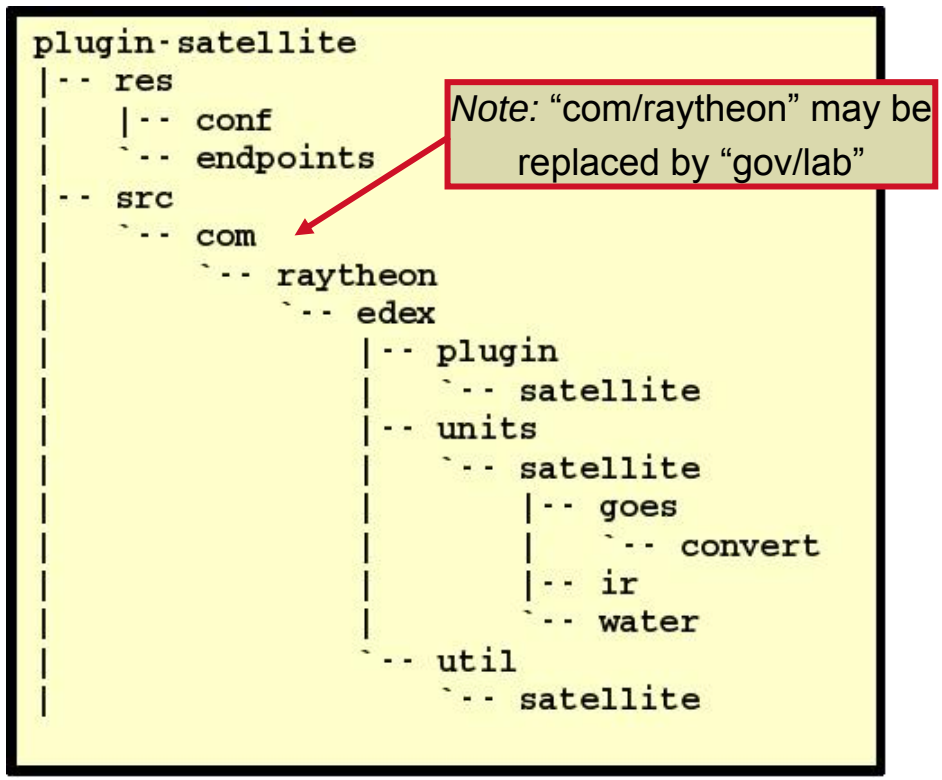
- EDEX Database pooling utilizes the generic object pooling API provided by the Apache Commons Pool project
- More information available at: http://commons.apache.org/pool/

Questions?

Revised Database Definition Pattern

EDEX Plug-Ins: The Plug-In Concept

- EDEX plug-in contains multiple classes and definition files
  - A sample plug-in directory structure – the satellite decoder plug-in – shown here
  - A plug-in is not limited to this directory structure; it may contain additional directories
- EDEX services use the Plug-In to decode data, store data, retrieve data and convert data for transfer to clients



EDEX Plug-Ins: Plug-In Configuration Files

- Most aspects of an EDEX plug-in controlled by a set of files
  - Although not strictly required, most Plug-Ins have these files

Folder	File	Description
<root>	build-components.properties	Defines the build dependencies for the plug-in
	client-includes.dat	Defines the files to include in the client version of the plug-in
<root>/res	binding.xml	Defines the JiBX bindings for data classes in the plug-in
	<data-type>.db.xml	Defines the database table definitions for the plug-in
	<data-type.hbm.xml	Defines the Hibernate mappings for the plug-in
<root>/res/conf	plugin.xml	Defines run-time configuration values for the plug-in
<root>/res/endpoints	<name>-ingest.xml	Defines the Ingest ESB definitions for using the plug-in for data ingest

Note:

- <name> is the plugin name, e.g., grib
- <root> is plugin-<name>, e.g., plugin-grib

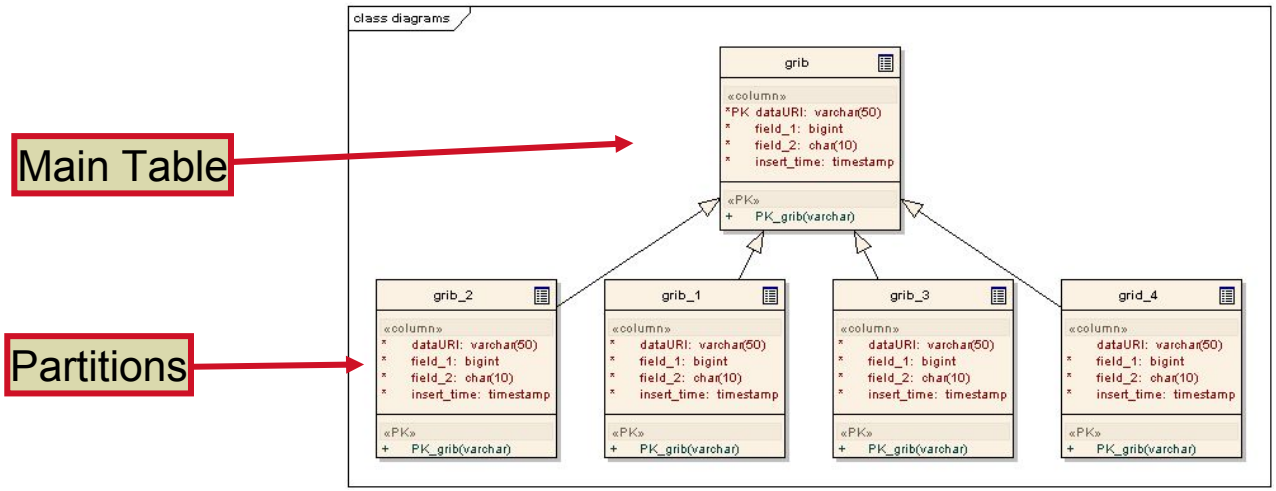
EDEX Plug-Ins: DB Table Generation

- EDEX has ability to generate Plug-In related database tables automatically
  - Table generation controlled by the Plug-In's <data-type>.db.xml files
  - Each file contains the definition of a single database file
- A Plug-In's tables are generated when the Plug-In is first added to the EDEX system
  - Tables are also added following a complete database purge

EDEX Plug-Ins: Data Retention

- EDEX uses the data table definition file to specify data retention for automatic data management
  - Maintenance consists of deleting old data files (HDF5) and purging old data from the database (PostgreSQL)
  - Data maintenance is performed
    - When the EDEX server initially starts
    - At 15 minutes past the hour as EDEX continues to execute
- EDEX uses a partitioned approach to data management
  - Data partitioned into 4 segments
    - In the database, each data table has 4 partitions
  - Data deletion accomplished by deleting/purging appropriate partition
- Example: If a retention time of 24 hours is specified, each partition will contain 12 hours of data
  - More on this later

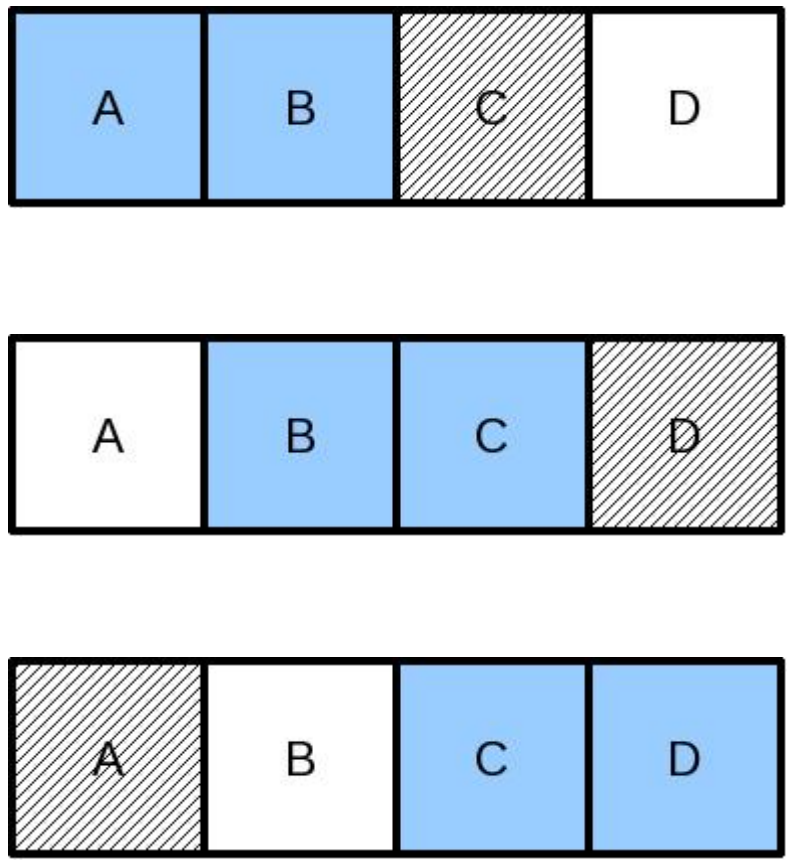
EDEX Plug-Ins: Data Table Partitions



- EDEX database uses PostgreSQL partitioning of database tables
  - Each table has 4 partitions
  - Each time data is purged; the oldest partition is deleted and re-created
- Partition usage (once the server is running)
  - Two partitions contain archived data
  - One partition is being actively used to store new data
  - One partition is being deleted and re-created

EDEX Plug-In: Data Table Partition Example

- Assuming a specified retention time of 24 hours
  - Each partition will hold 12 hours of data
- Data stored into partition A & B, D as a "spare"
  - After 24 hours, data begins to store into C
  - A minimum of 24 hours data is retained
- When C is filled, data is stored into D
  - At that point, A is purged and re-created
  - Once again, a minimum of 24 hours
- When D is filled, data is stored into A
  - At that point, B is purged and re-created
- This process continues as long as EDEX continues to execute!



EDEX Plug-Ins: DB Table Definition

```
<tableDefinition>
    <tableName>grib</tableName>
    <hibClass>com.raytheon.edex.plugin.grib.GribRecord</hibClass>
    <retentionHours>24</retentionHours>
    <order>1</order>
    <defaultClass>true</defaultClass>
    <partitioned>true</partitioned>
    <linkedExternally>true</linkedExternally>

    <columnDefinition>
        <name>datauri</name>
        <columnType>varchar</columnType>
        <constraintType>PRIMARY KEY</constraintType>
        <precision>512</precision>
        <scale>0</scale>
        <dataURI>false</dataURI>
        <index>none</index>
    </columnDefinition>

     ...

</tableDefinition>
```

Example of a db.xml file. The next few slides explain each element.

EDEX Plug-Ins: DB Table Definition (cont'd)

Tag	Description
tableDefinition	Defines a single table. Each table definition file contains a single tableDefinition tag set
tableName	Defines the name of the database table to create
hibClass	Defines the name of the class that represents a record in this table
retentionHours	Defines the number of hours to retain data in this table
order	Defines the order in which related tables are created
defaultClass	Specifies if this table is the default (base) table for the plug-in
partitioned	Specifies whether the table may be partitioned
linkedExternally	Specifies whether entries in this table are linked to data in the HDF5 repository
columnDefinition	Defines a column in the table. A each table contains multiple column definitions

EDEX Plug-Ins: DB Table Definition – Additional Information on Table Definition

Raytheon

- *order* tag: A Plug-In may specify multiple tables, each specification in a separate file. The order tag determines the order in which tables are created. Each table in a Plug-In must have a different order value. This allows a Plug-In to:
  - Specify multiple tables and
  - Include dependencies between the tables.
- *defaultClass* tag: A Plug-In may define multiple tables for its data. The defaultClass tag specifies if the table is the main for the Plug-In. This table is the one used by term query. Exactly one table may be designated as the default.

EDEX Plug-Ins: DB Column Definition

Tag	Description
columnDefinition	Defines a single column of the table
name	The name of the column
columnType	The column type
constraintType	Constraint type for the column. Valid values are PRIMARY KEY, UNIQUE, FOREIGN KEY, NONE
precision	The size, number of characters or digits, of the field
scale	(For numeric floats only) the number of decimal places
dataURI	Specifies whether this column is used in creating the data DUI
index	Specifies whether this column is indexed. Valid values are single, composite, none

EDEX Plug-Ins: DB Column Definition – Data URI Column

Raytheon

- Main table for a data-type plug-in includes a data URI column
  - The data URI column is the primary key for this table.
  - For externally linked tables, the data URI column links the meta-data in the table to the binary data in the HDF5 repository.
  - Selected columns in the table contribute to the data URI. These columns must be selected to provide a unique column value.

```
<columnDefinition>
    <name>datauri</name>
    <columnType>varchar</columnType>
    <constraintType>PRIMARY KEY</constraintType>
    <precision>256</precision>
    <index>none</index>
</columnDefinition>
```

Questions?

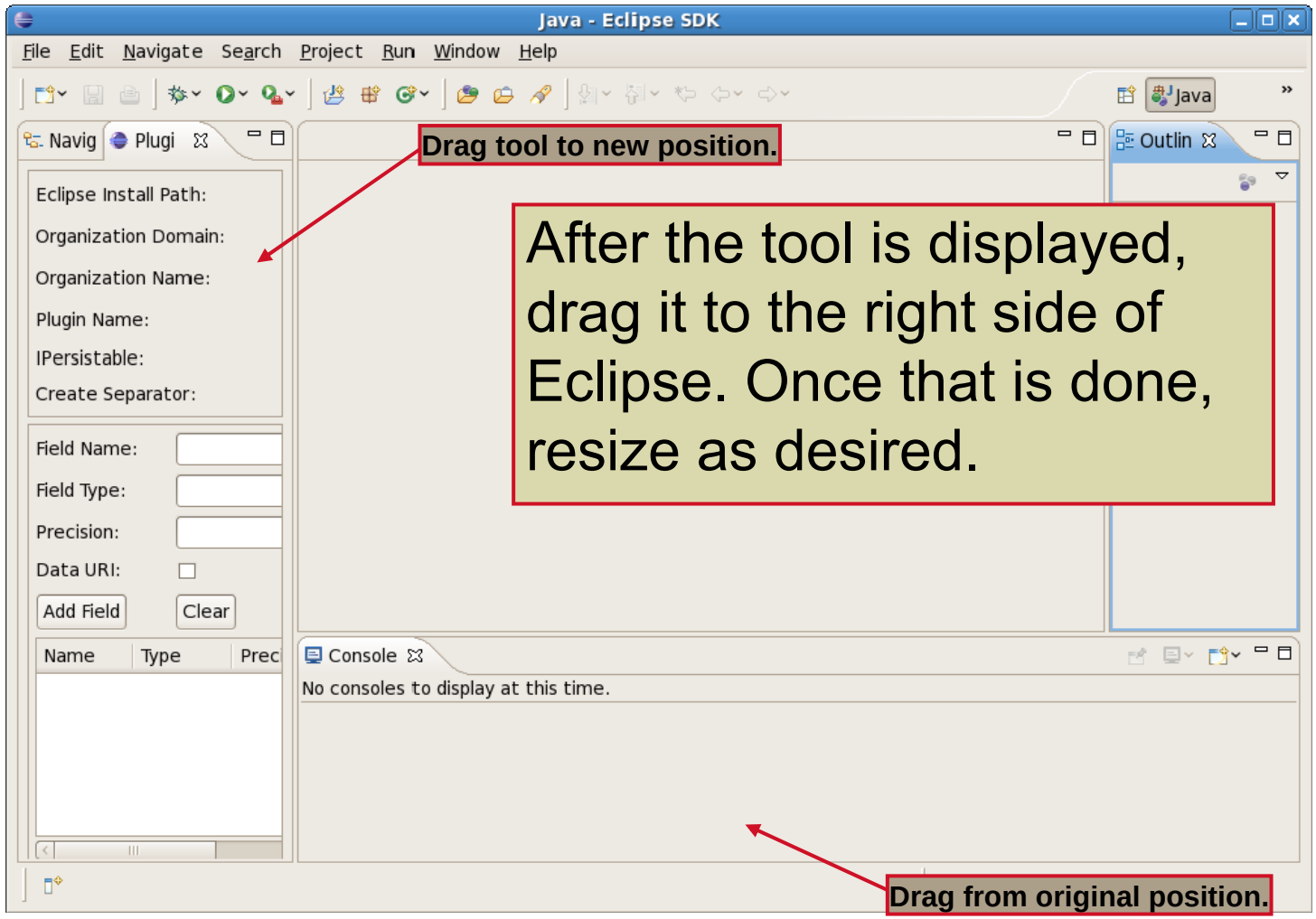

Innovation • Partnership • Responsibility
The Raytheon Missile Team

Tool Modifications for TO8

Problems Identified/Addressed

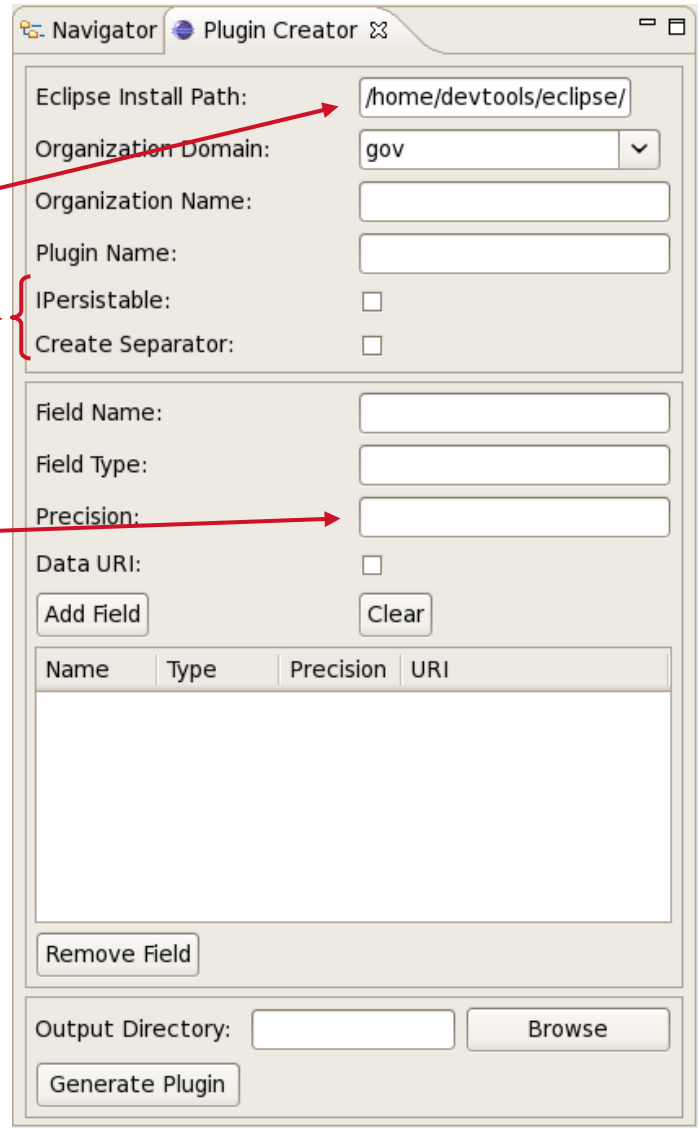
- Problems with the original Plug-In Tool identified by NWS:
  1. A need to better handle the “package name space”
  2. A need for scroll bars to be able to use the entire tool
- In addition, there was some refactoring of the Plug-In pattern
  3. A need update the tool to match the refactored Plug-In
- Of these,
  - #1 and #3 addressed via code changes
  - #2 is a consequence of Eclipse. We present a partial workaround. At this point, there is no way to use the tool on a “small” screen.

Displaying Plug-In Tool

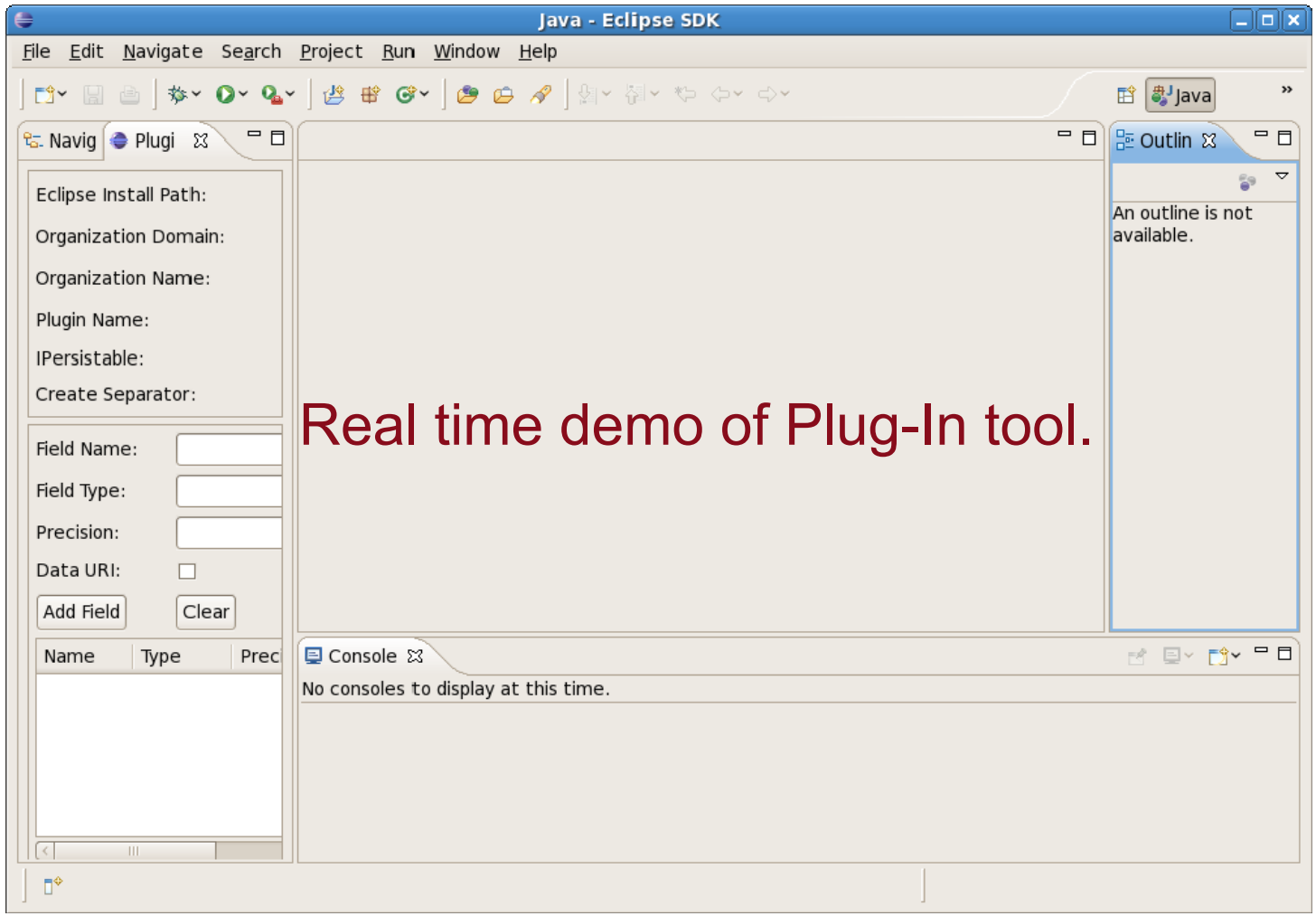


Plug-In Tool, New Features

- The Plug-In tool has been modified slightly to better fit thePlug-In creation pattern.
  - Code templates are part of the Plug-In, so you need to specify your Eclipse location.
  - Wording of checkbox labels has changed – does not change functionality.
  - A precision value has been added to the section for defining data entries. This is used mainly for defining database fields for strings. It should be set to zero (0) for other data types.



Plug-In Tool, Demonstration



Questions?


Innovation • Partnership • Responsibility
The Raytheon Mark's Team

Code Example

Code Walk-Through

Examination of Code for a Data-Type Plug-In Using Eclipse

Installer Modifications for TO8

AWIPS II Installer

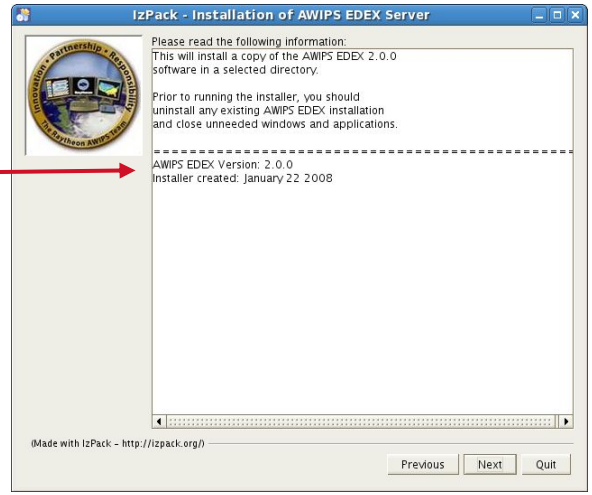
- With AWIPS II, there are now 3 installers
  - CAVE installer
  - EDEX installer
  - ADE installer

- AWIPS II Installers require Java

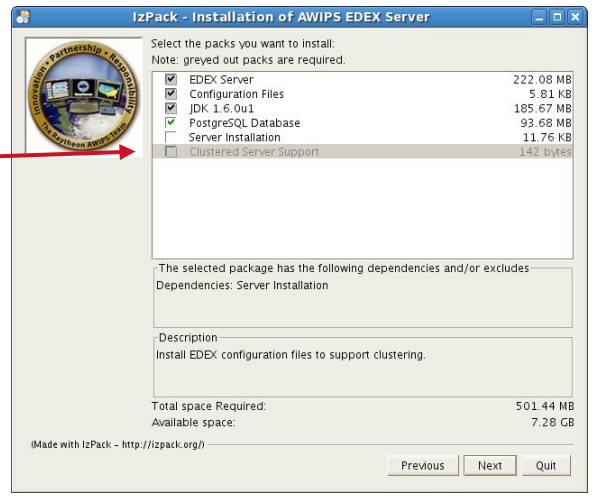
Note: *Support for EDEX installer on Windows has been dropped with T08.*

AWIPS II Installer Updates

- Versions have been updated
  - Shown here is the general information page, also on other pages

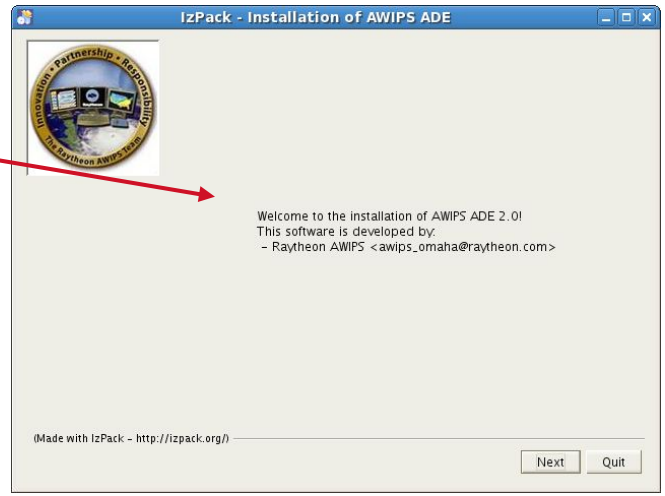


- Pack selection page has
  - Fewer choices
  - Support for clustered servers

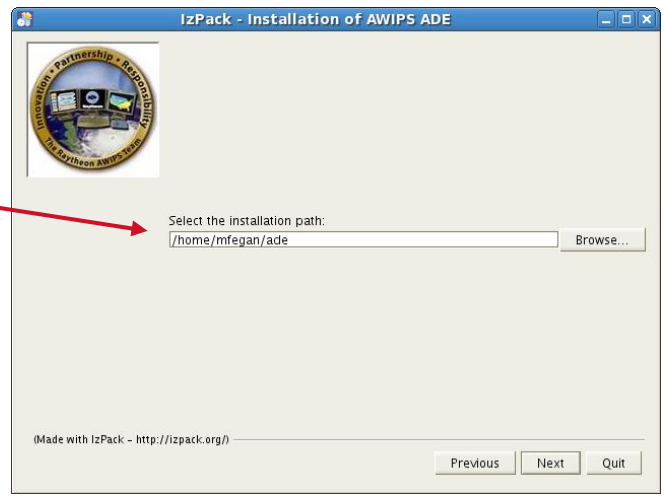


ADE Installer

- AWIPS II ADE 2.0 now has a separate installer
- Installs development tools
  - AWIPS II (TO 8) code base
  - Eclipse Europa (3.3.0)

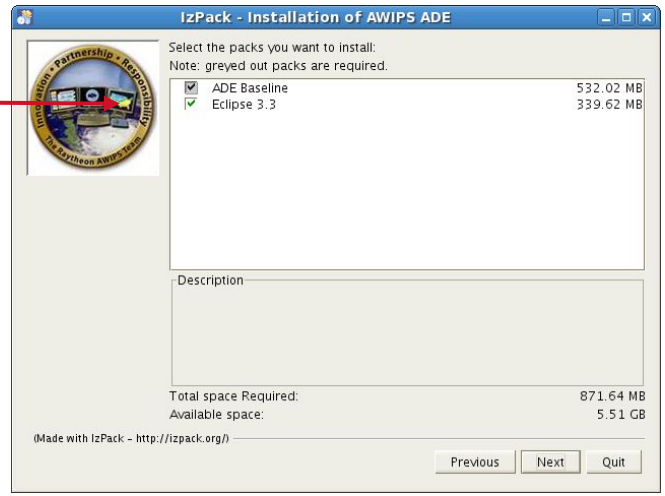


- ADE installation location is selectable at install time
  - Default location is “~/ade”



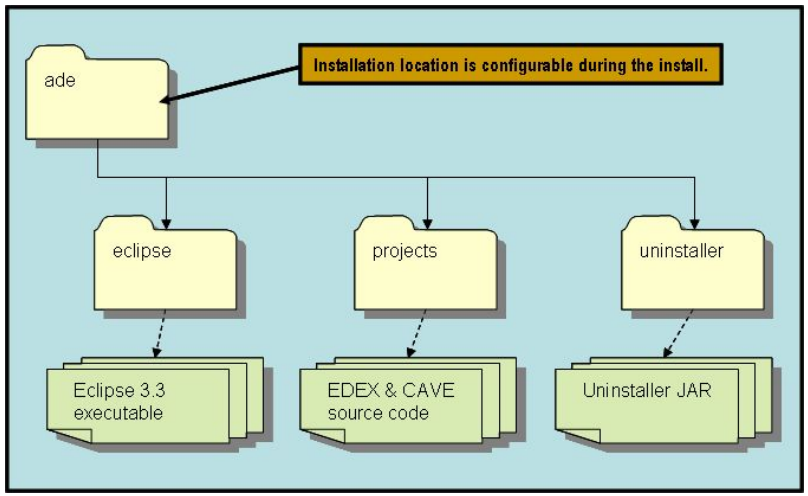
ADE Installer (cont'd)

- ADE components installed are selectable
  - Must install code base
  - Eclipse installation is optional
    - Eclipse install required for CAVE work



- Uninstaller provided as part of the ADE installation

ADE Installer (cont'd)



- Basic ADE 2.0 install structure shown here
  - Name and location of the root directory is configurable (at installation)
  - Default installation location:

    <user-home>/ade (Linux)

    C:\Program Files\ade (Windows)

Questions?


Innovation • Partnership • Responsibility
The Raytheon Missile Team

CAVE Modifications for TO8

Other Items of Interest

CAVE Modifications for TO8

- TO8 implemented selected AWIPS I functionality in CAVE
  - For the developer, this amounts to an addition of code to the baseline
  - The basic patterns and techniques used by the CAVE developer have not changed – refer to existing AWIPS II modules
- Any questions?

Coming in TO9

- Additional AWIPS I functionality
  - Core GFE
  - AVN FPS
  - D2D functionality
- Python Scripting support
- Local application support

Coming in TO T1 Training Updates

- Documentation of existing uEngine Tasks
- Updates to existing Training Modules (1 – 11)
- Annotated code examples

Questions?

Wrap-Up

Summary

- Covered modifications in EDEX dataflow introduced in TO8
- Covered modifications to the Data Access Layer Implementation introduced in TO8
- Covered modifications to the Database Definition Pattern introduced in TO8
- Covered modifications to the to the Plug-in Creation Utility introduced in TO8
- Covered modifications to the Installer for TO8

Resources

- On the ADE 1.0 DVD (TO8)
  - Current code available for examination in the CAVE baseline
  - JavaDoc documentation available