6.2 OVERVIEW OF THE NEW AWIPS SOA

John D. Lawson, Frank P. Griffith*
Raytheon Company

Jason P. Tuell, Ronla K. Henry
NOAA/National Weather Service

1. INTRODUCTION

The National Weather Service (NWS) Advanced Weather Information Processing System (AWIPS) is undergoing an extensive re-architecture and conversion to a Services Oriented Architecture (SOA). A primary goal of the new architecture is to provide a software environment that meets the NWS continually growing mission requirements. This requires an adaptable system able to readily incorporate new data types and new science applications while providing scalability from a laptop computer to high performance clustered servers.

Raytheon Company is currently developing the new AWIPS SOA. The technical reference architecture is an executable environment that provides an “end-to-end” AWIPS development environment (ADE) and runtime environment created by automated build and deploy processes.

This paper provides an overview of the conceptual architecture and design approaches being used. The paper also discusses the features and benefits provided by the new architecture.

2. KEY REQUIREMENTS / NEEDS

The existing end-user functionality of AWIPS appears to be comprehensive and adequate for current needs. Of concern to the NWS, however, are the cost, complexity, and rapidly increasing difficulty of extending AWIPS’ functionality to meet the future mission of NWS and adapt to an evolving Concept of Operations.

The new Architecture addresses several AWIPS system needs identified by the NWS. These needs / requirements include:

- Improved adaptability to accommodate new science, new data types, and a changing CONOPS.
- Maximum use of Open Source software
- Platform independence (hardware, operating system, database).
- Improved reliability, availability, and supportability.
- Improved performance, scalability (up and down), and load balancing.
- Simpler software build and deployment framework.
- Streamlined installation process, including application releases.
- Improved support for local applications in site installations.
- Standard development environment for all developers and improved compliance with standards.

3. APPROACH

The project approach of this software architecture conversion first creates a software infrastructure and then utilizes that infrastructure to migrate end-user functionality (e.g. applications) to a new execution environment. The design approach creates a new low cost framework for hosting a full range of environmental services. The framework scales down to a small laptop and up to clusters of enterprise servers without software change. The framework is based on highly reusable design patterns that maximize reuse, have data type independence, and fast adaptability.

Reuse is maximized through leveraging open source projects which also meets the NWS desire for non-proprietary software usage.

The framework implementation integrates several best of breed open source projects with a set of advanced enterprise patterns to create a highly extendable framework. The open source codes and enterprise patterns are packaged into the “AWIPS Development Environment” (ADE) that

∗ *Corresponding author address*: Frank P. Griffith, Raytheon Company; email: Frank_P_Griffith@raytheon.com

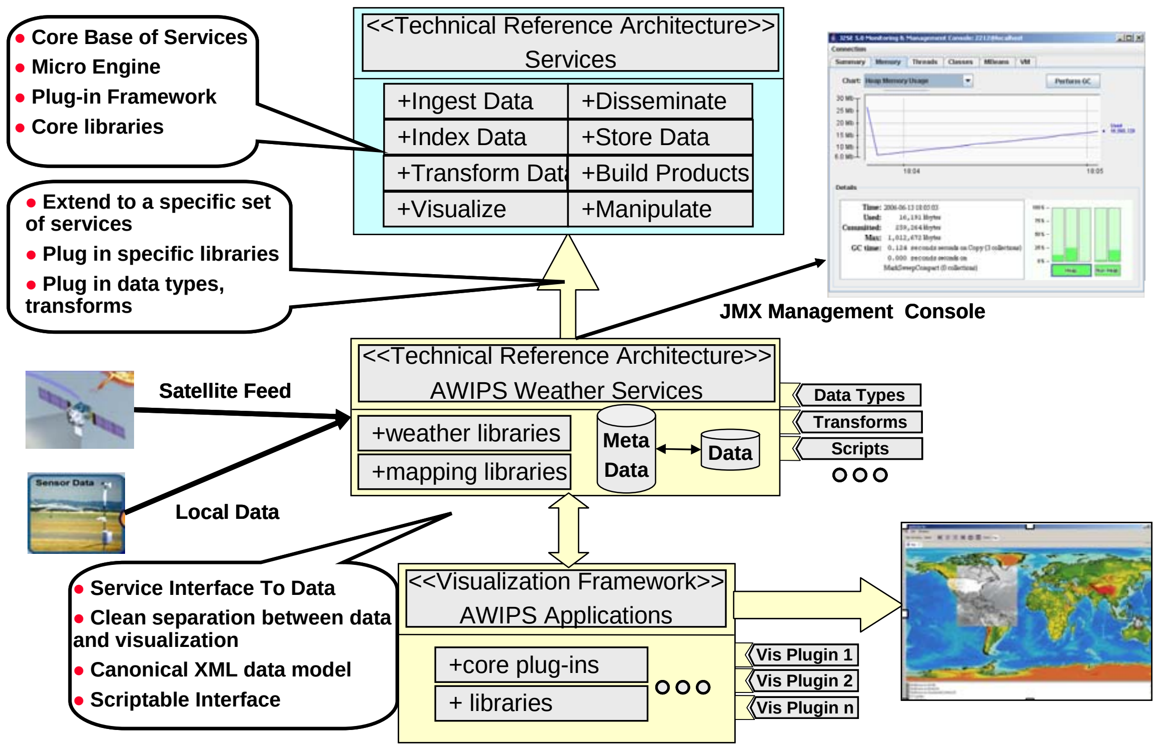

Figure 1 - AWIPS SOA Technical Reference Architecture

includes everything from the source code repository to the execution environment including operator clients.

The technical reference architecture, shown in Figure 1, is an executable environment that provides an “end-to-end” AWIPS development environment (ADE) and runtime environment created by automated build and deploy processes.

One important aspect of the architecture is that the core services are extendable to varying implementations. As shown in the figure the core services are extended to the specific needs of the AWIPS Weather Services. Other instances of this architecture could include different libraries and metadata for other application domains. Another important aspect of the architecture is the separation of the data and the visualization framework which ensures loose coupling of data and the visualization applications. Loose coupling enables system flexibility and adaptability.

4. ARCHITECTURE FEATURES

SOA has become a buzz word of sorts. In simple terms, what SOA means to AWIPS is:

- Services are organized into containers with loose coupling.
- Services are composed of components
- Interface details are abstracted away from services.
- Interfaces between services and clients of services are defined in a well-known data model.
- System capabilities are available as network services.
- Service are event-driven

Additional features are discussed in the following paragraphs.

Languages and Inter-process communication

Java is the primary programming language. Java has extensive open source support, high programmer productivity, high reuse, performance parity with traditional languages, and is a primary university teaching language enabling a population of future Java programmers. Java also enables platform independence via the Java Virtual Machine (JVM).

Python is supported with a Jython engine for scripting. The Jython engine is extensible with Java classes, has a large base of customer scripts and expertise, and provides a clean OO approach to scripting

The Java Messaging System (JMS) is the primary mechanism for inter-process communication. JMS enables Staged Event Driven Architecture (SEDA) processing, increases reliability through queue persistence, and enables subscription / notification through topics. JMS also enables asynchronous communication for improved performance.

HTTP, FTP, JMS, e-mail are utilized for WAN communication. Transparent switching between JMS and HTTP occurs without application changes.

Execution Container & Data

The new AWIPS SOA utilizes an Enterprise Service Bus (ESB) approach to integration that provides the “plumbing” for highly distributed loosely coupled services. The ESB supports messaging, web services, data transformation, and routing. Process flow and service invocation can span the entire bus. A dependency injection container minimizes service and component coupling enabling more flexible services. The ESB provides clear separation between business and control logic.

Visualization

A Common AWIPS Visualization Environment (CAVE) is provided to facilitate consistent user interface implementations across varying AWIPS applications. CAVE supports the fixed scales and detailed interactions of D2D and also supports the large data sets and analysis capability of N-AWIPS. CAVE supports GIS Visualization natively as well as collaboration and remote client operations.

CAVE is implemented as a set of Eclipse Plug-ins installed in the Eclipse Rich Client Platform (RCP). Eclipse RCP is a full featured framework with an extensive widget set, extendable through plug-ins. It has high performance and is available as open source.

Eclipse RCP has an extensive support community, and a large public repository of plug-ins.

Geospatial Enabling Data

All ingested data is indexed into a spatial index making spatial query and analysis available to the visualization operator or SOA service. Station and scale definitions are converted into ESRI standard shape files. An R-Tree index of geometry records is created. (A prototype using 7000 record station list had a query performance of ~ 230 spatial queries / sec on a laptop). GIS analysis of data and rendering GeoTiff using Tiff tags are enabled. The system can ingest and index data spatially and create Shape and GeoTiff output. ESRI Shape Files are used as the standard vector format. This overall approach is free, simple, provides high performance, and is based on an industry standard data format.

Extensibility

The new AWIPS SOA enables adaptability and flexibility by providing several mechanisms to extend to extend the system’s functionality either as a centrally or locally developed capability. New data types and transforms are readily added to AWIPS using plug-ins. New visualization capability is added to CAVE using plug-in support provided by the Eclipse RCP. The uEngine provides an execution framework for generating custom products on-demand. Customer systems can request products by script requests over a network. The script performs small general units of work that get chained together to produce a customer product. Other examples of extensibility are support for GEMPAK style grid diagnostics and GFE style scripting using Python.

Patterns and Services

Several design patterns have been implemented in the new AWIPS SOA to maximize reuse across system functions. A list of example patterns is shown in Figure 2 on the next page.

Patterns and components are combined to form various services. Sample services of the new AWIPS SOA are shown in Figure 3 on the next page.

Performance

Quad tree tiling and progressive disclosure enable high performance graphical display pan and zoom on relatively modest software for vector and raster data.

Staged Event Driven Architecture (SEDA) is employed with multithreading and clustering to provide scalable performance without software changes.

The new AWIPS SOA described in this paper provides NOAA/NWS with a modern software

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 Plugin Pattern	Enables system adaptability to new data and transforms
Legacy Adapter Pattern	Enables system evolution by allowing legacy processes to run in a SOA
Data Notification / Subscription	Enables data driven processing and display
Common AWIPS Visualization Environment (CAVE)	Consolidates disparate display mechanisms into one platform independent whole

Figure 2 – Sample Design Patterns Focused on Maximizing Reuse Across System Functions

Service	**Description**
Ingest	Listens on an endpoint for new data and transforms the data into a message.
Persist	Writes ingested data to a persistent store; file system or RDBMS.
Index	Indexes the metadata extracted from the ingested data into a store that facilities data searches and retrievals.
Product	Listens on an endpoint for external product requests and fulfills the request with a response message. Typically receives “Action” scripts that describe how to transform raw data into a visualization product.
Notify	Broadcasts a product from a subscription fulfillment. Also, sends out alerts based on ingested data.
Subscribe	Listens on an endpoint for URI references to new data and determines which products need to be rebuilt.
Auto Build	Receives requests to build products that are under subscription. Triggered by data arrival and/or time.
Collaborate	Provides a common point for serving out products that are shared by several clients.
Grid Edit	Receives requests to edit grids. Interfaces to a scripting engine to allow edit requests to be scripted.
Adapter	Enables legacy command line programs to be run as a standard service.

Figure 3 – Sample Services of the New AWIPS SOA

architecture that should provide the adaptability needed to meet future missions for many years.