Software Architecture Engagement Summary

Presented to:

George Smith, Chief, Hydrology Laboratory (HL)
Jon Roe, Chief, Hydrologic Software Engineering Branch (HSEB)
Edwin Welles, Hydrologist, Hydrologic Software Engineering Branch (HSEB)

January 14, 2004

APEX DIGITAL SYSTEMS

Introduction

In early November 2003, the Office of Hydrologic Development (OHD) of the National Weather Service (NWS) engaged Apex to collaborate with OHD in defining new architectural capabilities provided by recent advances in software architecture. The capabilities are relevant for OHD's efforts in renewing its existing forecasting software architecture, and defining a design path for a gradual process of replacing current system components.

The objectives of Apex's engagement were defined as follows:

- Present to key OHD staff architectural concepts that are at the core of new major application management models (J2EE, .NET) that may be used as the basis for next development stages for OHD software.
- Discuss with key OHD staff overarching architectural priorities Apex considered critical for future OHD systems, and which were also implemented in the proof-of-concept Workflow Management System (WMS) .
- Discuss high-level future OHD system requirements, and how the major application management models may address them.
- Discuss other approaches that may diverge from the major models but may be valuable in meeting OHD system requirements.
- Discuss OHD expectations of distributed data and algorithmic services.

The overall result from the engagement was to define a common terminology and understanding of the new architectural and application management concepts and capabilities that exist in the J2EE and .NET architectures, and how these might apply to a next generation OHD application environment.

The engagement was structured into three meetings between George Smith, Jon Roe and Edwin Welles from OHD and Manuel Mattke and Stephanie Liu from Apex. The first meeting focused primarily on defining the concept of a Services Oriented Architecture (SOA) as well as on clarifying terminology. The second meeting focused primarily on a range of architectural themes that heavily influence the selection and design of a future architecture for OHD systems. The third and final meeting consisted of discussion of a specific conceptual architecture plan designed and proposed by Apex based on the foregoing discussions.

This document represents one of the deliverables under the engagement, intended to summarize key concepts. In addition to this document, Apex has or will deliver several schematic architecture plans, additional reading material and a glossary of terms, both of which are attached to this document as appendices.

Services Oriented Architecture

During the first meeting under the engagement, the participants discussed extensively the concept of a services oriented architecture (SOA), how it differs from traditional component-based architectures (such as COM or CORBA), and how it extends the notion of effective componentization of capabilities to a new level.

OHD has articulated several key architectural priorities for a future software architecture, including:

- Continued functional stability and reliability at current or higher performance levels.
- Concurrent processing of data on multiple platforms, potentially in different locations.
- Significantly improved data management and record locking, including a gradual transition to using standard relational database management engines for all data.
- Gradual transition away from FORTRAN, using more current languages such as C++, Java, and others as replacements.
- More flexible application configuration and management to replace the current release model, which requires a complete build by OHD's software engineering branch for each system release.
- An increased ability to incrementally evolve the architecture in a way that matches on-going budget realities.
- The ability to integrate the wider hydrology community, customers and collaborators from various contexts.

While the more traditional component object design approaches such as COM or CORBA would address many of the requirements described above, these architectures rely either on software builds similar to the builds produced today through full compilation of source code, or on remote procedure calls and method invocation such as DCOM or RMI that are relatively inflexible and plagued with concerns about stability and performance.

As an alternative, a Services Oriented Architecture based on web services for synchronous data exchange and on messaging for asynchronous data exchange holds significant promise for OHD's steps in upgrading its software systems. Most importantly, systems built on a Services Oriented Architecture provide a high degree of configuration and management flexibility because they no longer are compiled into one release. Instead, such systems rely heavily on independent services published in any web-accessible location, together with directory entries that define the locations to be used for calls to such services.

In addition, a Services Oriented Architecture enables gradual, piece-by-piece replacement of key software components without requiring a complete overhaul of the entire OHD software environment, a key requirement listed above.

Today, the two major providers of SOA-based application frameworks are Microsoft's .NET framework and the open-source J2EE standard. Both offer a variety of important capabilities, such as:

- Transaction management
- Scalability
- Security
- State management
- Application integration services
- Administration services
- Run-time services
- Connection services
- Messaging services
- Application development, deployment, and execution platform
- Web services
- Business process management services
- Support for various graphical user interfaces, including web-browsers and wireless devices

Given OHD's Linux-centric approach, the J2EE (Java 2 Platform Enterprise Edition) standard appears to be more suitable, even though it appears that the distinction between the frameworks may become less relevant in the future as they begin to support each other's features. For example, JMS (Java Message Service) traffic can be processed by .NET queues, and Java components fit seamlessly into either architecture without modification.

Architectural Themes

During the second meeting of the engagement, Apex defined a variety of high-level themes relevant to the architecture project. These themes cover both process priorities and technical priorities, which should all be addressed in some way as part of the gradual implementation of a new architecture. The themes outlined here already assume a services oriented implementation.

A. Process themes:

- **Design priorities,** which will fundamentally define the architecture for any future implementation effort:
    - o <u>Distributed Modeling:</u> as OHD moves to a distributed forecasting process in the context of the national digital forecast database, multi-threaded execution, location independence and other themes will become critical design drivers.
    - o <u>Usage Patterns:</u> currently, the three apparent usage models are automated background operation, automated operation in response to a remote request, and manual operation based on a local user-interface request. Additional usage patterns may become apparent in the future.

  - Common Baseline Structures: as OHD's software capabilities increasingly become available to non-NOAA users, defining a standard, self-describing ontology will become more important. In addition, supporting a community process around the development of open source software at OHD will be important to enable enhanced interaction with such non-NOAA users without significant expansion of OHD customer services resources.

- **Migration Process priorities,** which will shape specific implementation capabilities and approaches:
  - Functional Chunking: determination of the size of component/service.
  - Transition Strategies: how to thread in new algorithms and data providers.
  - Hardware Considerations: determination of the appropriate hardware and where it should/could be located.
  - Operating System Considerations: determination of how dependent the new implementation will be on operating system-specific capabilities.
  - User Interface Integration: determination of preferred UI implementation, using web-based interfaces, fast compiled clients or java-based applet clients.
  - Legacy Code Wrapping Strategies: FORTRAN will be relevant for OHD systems for some time, requiring a determination of a standard wrapping approach.

- **Partners** who will add capabilities or support to the upgrade process, or will impose restrictions on it:
  - National Academy of Science
  - NOAA Leadership
  - NWS Leadership
  - OHD teams
  - AWIPS engineering teams
  - River Forecast Centers
  - Weather Forecast Offices
  - Other NOAA groups
  - Other governmental agencies
  - Academic Hydrologists
  - Commercial Hydrology Software Companies

While important, the process themes above were addressed only briefly. Their relevance will increase as specific priorities emerge from the technical themes and as implementation strategies become apparent.

B. Technical Themes:

The technically related themes focused on key parts of a Services Oriented Architecture. Combined with the framework of an application server based on the J2EE or .NET standards, these technologies provide a full services-based run-time environment. Each of the themes listed here will be discussed more fully in separate sections below:

- **Algorithm Services**, which provide computational capabilities
- **Display Applications**, which provide user interactivity and visual analytic capabilities
- **Control Services**, which manage the overall system operation, including:
  - Directory Services: provide data on the physical or logical location of services
  - Workflow Services: provide specific execution scheduling, sequencing and load management capabilities
  - Logging and Diagnostic Services: provide capture capabilities for error and program run-time diagnostic information
  - Security Services: provide system security and authentication capabilities.
- **Data Services**, which provide all data to data consumers throughout the system and beyond. For data services, several architectural features will be important:
  - Data location management
  - Data format management

- o Data repository location transparency
- o Native data source management.

The fact that each of the themes or parts is described as a "Service" is indicative of the nature of a Services Oriented Architecture: all aspects of the environment are designed as self-standing components with their own interface structure to allow flexible application configuration at run-time based on standard protocols such as SOAP (Simple Object Access Protocol). **With this approach, an application consists of a small, compiled application structure that makes calls to services for algorithms, display, security, control flow and data.**

In the following section, each of the services will be described in further detail.

Algorithm Services

Algorithm Services will be a collection of computational capabilities that other applications can call upon. A calling application would provide the data set required for a particular algorithm, and the algorithm will return a specific return value or a return data set containing the computed information.

Each algorithm service should be designed to provide specific, relatively atomic computational capability to avoid code overlaps with other algorithm services. Instead of overlapping code, multiple algorithmic services should be implemented that use each other's capabilities.

Algorithm services will typically be accessed via web services for synchronous data exchange. For background operation, asynchronous access via messaging could also be provided.

Early in the transition from the existing OHD architecture to a Services Oriented Architecture, most existing applications will be implemented to algorithm services with few modifications. This will make the existing code base available in the new architecture, and will stabilize the new architecture until the point where time and resources permit further modification. Such modification would then consist of breaking apart an existing application into various computational components, each of which would be deployed as a self-standing algorithm service.

A critical aspect of service-based algorithms is that the algorithms become interchangeable. For example, RFC software developers will be able to write and deploy alternative algorithms for use without requiring a new system build, simply by locating the service on a web-accessible server running the application framework and updating the services directory with appropriate information.

While Data Services will provide persistent data storage (see below), each algorithm service will usually require its own temporary data storage capability to store data during computation. While local database engines may be used for such purposes, the typical approach to such temporary data storage will likely be file-based XML.

Display Applications

Display applications are the future equivalent of current "GUI Applications". Such applications are managed by users in real-time, and typically process and display data on-screen. A key requirement for these applications is processing speed, as the user typically waits for the results of computation and display before making further decisions.

Currently, GUI applications are implemented in C, C++ or Java, and future display applications would likely be implemented using the same languages. However, instead of encapsulating all code required to perform the needed calculations, the applications will use Directory Services (see below) to determine which algorithm service to call on for specific computations. Once the algorithm service is located via the directory service, the display application then calls the service directly and provides any needed data to it. When the service returns the computed values, the application can then display the results.

The proposed architecture will also require several special-purpose display applications, such as a diagnostic and logging application and a workflow console. The diagnostic and logging application will display log and run-time data for a specific application or workflow involving several applications. Users will still be able to review local log files, but will also have the options of viewing log and diagnostic information in the context of an entire process involving multiple applications. This capability could also be deployed as a web-based interface for remote access to diagnostic data.

The workflow console will enable users to construct, save and execute workflows, which are sequences of application calls that produce a particular product or computational result. Both the workflow console and the diagnostic and logging application will use data services to locate and extract the data needed for display to the user.

Control Services

A fundamental architectural aspect of the Services Oriented Architecture is the fact that application control structures become explicit, after being implicit in a compiled application. Now, application flow must be defined, and directory information will provide run-time configuration and location data. Each of the control services listed above are discussed in more detail below.

Directory Services

Directory Services provide the logical location for a particular algorithm or data service, or for backup directory services to a calling application. Conceivably, each application could include the logical location for the data and algorithm services it requires, resulting in a portable, but non-configurable application. To ensure run-time configurability, each application should only know unique descriptors for the data and algorithms it requires, and depend on a local or remote directory to resolve the actual name and location of the service.

For both web services and message traffic in the J2EE environment, the location returned by the directory service will be a URL. Directory services may also provide information on required parameters, even though such information could be derived using UDDI (Universal Description, Discovery, and Integration). Depending on architectural and performance requirements, directory services may cache UDDI data and refresh their information periodically.

Similar to the Domain Name Servers (DNS) that provide directory information for the Internet, directory services should be implemented and managed in a redundant fashion: each workstation or server that functions as a node in a services oriented system should have access to multiple, redundant directory services that are processed in a defined order. With this approach, run-time configuration can occur at the local directory, with all default values provided by a remote directory service.

While OHD may want to implement a proprietary directory services environment, existing LDAP applications could be used to provide the needed information.

Workflow Services

For purposes of this discussion, a workflow is defined as a sequence of applications that are executed in a defined order. The order may be defined simply by a scripted sequence, or by events that occur during the execution of the applications. E.g., one application's exit condition may define the following application's input parameter range.

Workflow services provide the user with the capability to define application execution sequences as well as to manage such sequences at or after runtime. Workflow services also provide a management infrastructure for applications running unattended as background jobs. This management infrastructure includes the ability to detect whether a scheduled application launched as requested, whether a running application is deadlocked or aborted with exceptions, or whether an entire workflow completed successfully.

To achieve this level of control, each workstation or server that participates in the overall runtime environment must run a workflow agent that reports the execution status and diagnostic data for each launched application back to workflow services.

In addition, workflow services will need to provide load management services by monitoring the load factor for each workstation or server available in the system. By monitoring in real-time the load status of any given machine, workflow services are capable of launching algorithm or data services on systems that have the most available bandwidth, resulting in pro-active performance management.

To achieve proper load balancing the workflow agent, or an equivalent separate agent, must provide load status data to workflow services on a regular interval, to be determined by the user.

As service oriented components, workflow services enable fully distributed operation and real-time, lightweight access to execution control data from within other applications.

<u>Logging and Diagnostic Services</u>
In the current OHD software environment, each application manages its own log file for diagnostic and logging purposes. Logging and diagnostic services will serve as a central, database-based collecting point for all diagnostic or log output from applications.

Part of the data collected by this set of services is used by the workflow services to determine application entry and exit status values that drive workflow status. E.g., if an application sends information about an abnormal termination to the logging and diagnostic services, workflow services will analyze this data and decide to also terminate the currently running workflow, unless the abnormal termination is irrelevant to the success of the entire run.

The diagnostic and logging display application described above will derive diagnostic data from this set of services based on user-specified criteria, and display them to the user. With this approach, users will be able to review more easily the interaction between an application and the specific performance implications of specific workflow design and setup.

<u>Security Services</u>
Security services have several key functions. Most fundamentally, each service or component that forms part of the architecture should be registered and authenticated as valid before it becomes accessible or can access other system components. Security services will manage registration and authentication, most likely involving certificates or component signatures for each service or component. As a result, anyone with access to the environment may create new services or components, but will have to register such services or components with security services to actually use such code.

In addition, security services will manage personalization for users throughout the system, involving ownership control over workflows, sharing data among users and carrying directory settings across workstations or servers.

Finally, security services will be able to enforce encryption requirements, where necessary. E.g., all web services or message traffic can be set to run over HTTPS, and security services will enforce cases in which system administrators or users request such security.

Data Services

Data Services likely represent the most complex aspect of the services oriented architecture. This group of services must provide data in real-time, from the correct source, locking the appropriate data spaces, and in the appropriate formats. As a result, the design for data services will need to incorporate:

- <u>Data location management:</u> since OHD will continue to use a variety of data sources in many formats for some time, managing directory information about where data resides must be universally available across the system. In some ways, data location management will be similar to the Internet's DNS records that correlate IP addresses with domain names.
- <u>Data format management:</u> given the variety of data sources and native formats, data services need to be able to provide data in any format requested by a calling application

regardless of the native storage format. E.g., while data may continue to be stored in FS5 files for some time, calling application may request that returned data be presented as serialized record set objects. Format management capabilities will ensure that data is transformed appropriately. In cases where no formatting changes are required, data format management components will simply pass the results through to the next higher architectural component layer.

- Transparent management of data exchange with local or remote data repositories: as the concurrent, multi-threaded architecture grows, it will need to ensure that data is available to callers regardless of the actual location of the data. When data is located via location management capabilities, a calling application must be able to communicate with the data provider without regard to the physical location of the data. The components that manage this data exchange will provide this important interface layer that will truly enable location-independent data access.
- Direct access management to native data storage behind services-oriented interfaces: Specific data components must be able to interface directly to the native storage formats such as Informix, PostgreSQL, FS5, XML or delimited text. Such access will occur using the current approaches such as direct connection strings, DSNs, pipes, and others.

Performance optimization of data delivery capabilities will require significant attention, with careful design of local data caching functionality to enhance throughput. Most likely, local data sources will be file-based XML, while remote storage will be implemented on relational database management engines such as PostgreSQL, Informix, and others. It would also be valuable to review the option of deploying PostgreSQL on all workstations or servers to create a more uniform operating environment, and to take advantage of the higher speed and stability of a true database engine. Further, the multi-user capable record locking features of a database engine will be important in the services oriented environment described here.

Additional Materials

The attached conceptual diagrams show the functional relationships among the services described in greater detail.

Conclusion

The proposed services oriented architecture offers significant benefit to OHD as it pursues a viable software architecture for future development. Apex has already demonstrated the functionality of key parts of the architecture, such as messaging among components using JMS (Java Message Service), and rapid integration of diagnostic and logging capabilities into existing applications.

Further evaluation will be needed before confirming the suitability of the services oriented architecture and defining a specific architecture design. Clearly, expanding the evaluation of algorithm and data services will be critical in showing that performance of display applications will be appropriate, and that data availability in a service mode will meet key functional requirements.