Building Open Spatial Resources Into GeoIntel Enterprise Architectures

Author: Chuck Heazel,
Open GIS Consortium, Inc.
cheazel@opengis.org

Introduction

Agencies are moving to Web Service-based architectures because they offer an economical, forward-looking way to integrate disparate applications within and between enterprises and improve information sharing within and between organizations. Enterprise information needs, however, are complex and changing and Web service architectures are evolving. To avoid implementation pitfalls, careful planning is necessary, with an eye toward unexpected shifts in the distributed computing environment.

> **Distributed Computing** - Distributed computing is a programming model in which processing occurs in many different places (or nodes) around a network. Processing can occur wherever it makes the most sense, whether that is on a server, Web site, personal computer, handheld device, or other smart device. *www.microsoft.com/net/basics/glossary.asp*

> **Web Services** - A Web Service is programmable application logic accessible using standard Internet protocols. Web Services combine the best aspects of component-based development and the Web. Like components, Web Services represent functionality that can be easily reused without knowing how the service is implemented. Unlike current component technologies that are accessed via proprietary protocols, Web Services are accessed via ubiquitous Web protocols (e.g.: HTTP) using universally accepted data formats (e.g.: XML) and standard "wrappers" (e.g.: SOAP).
> *Internet World*

Members of the Open GIS Consortium, Inc. (OGC) work together to develop open specifications for software interfaces and encodings that address interoperability issues involved in service-based geoprocessing. The uses of these specifications, and their relationships, are described in OGC's *OpenGIS Reference Model (ORM)*.

The Old Way and the New Way

The Web services computing model demands a different way of thinking about information systems. For many years, IT systems have been built using a version of the "waterfall" model, which has five steps:

1. Define the requirements.
2. Design the system.
3. Build the system.
4. Test the system.
5. Deploy the system.

Each step is performed in-turn, sometimes with feedback loops to assure that requirements, design, and implementation are consistent. The result is a stable implementation that meets the specified requirements. This approach, however, works poorly for web-enabled systems.

World Wide Web technologies make it possible to build systems that evolve and adapt to changing needs and capabilities. In these systems:

1. There is no single administrative authority or organization responsible for the system.
2. The business processes being supported by the system change, sometimes daily.
3. The system must support new and unanticipated business processes.

4. Capabilities can enter and leave the system at any time.
5. Technology can be inserted and removed from the system at any time.
6. The system must accept new and unanticipated information.

Waterfall based systems are not designed to handle the ambiguity and constant change that is characteristic of Web-enabled systems. Fortunately, the distributed object community, in particular the Object Management Group (OMG), has been developing techniques for designing systems with these characteristics for some time. The work in OGC has built on this foundation.

Framework, Transparency and Publish/Find/Bind

Creating coherent architectures in this chaotic environment requires a ***framework*** of constraints, processes, and patterns to guide component developers. This framework enables the integration of a broad range of anticipated work flows and technologies.

One key element is ***implementation transparency***, which mandates that each system component shall be treated as a "black box". All that a developer needs to know about a component is the description of its external interfaces. The consequences:

1) The system as a whole has no dependencies on the software implementation of a component. This allows a component to be developed, tested, and maintained as a stand-alone entity. If the implementation is true to the published interfaces, changes to a component have no impact on the system. Also, in an IT sector like the geoprocessing sector that has adopted a set of open published interfaces, it is probable that developers can choose from among multiple implementations of components with the same interfaces.
2) The system as a whole has no knowledge of the internal data structures of the component. All that is known are the models for information exchange described in the interface definitions. This frees the component developer to use whatever information management and storage techniques are appropriate for that component. Thus, the internal data format can be designed for processing efficiency, not end user requirements.

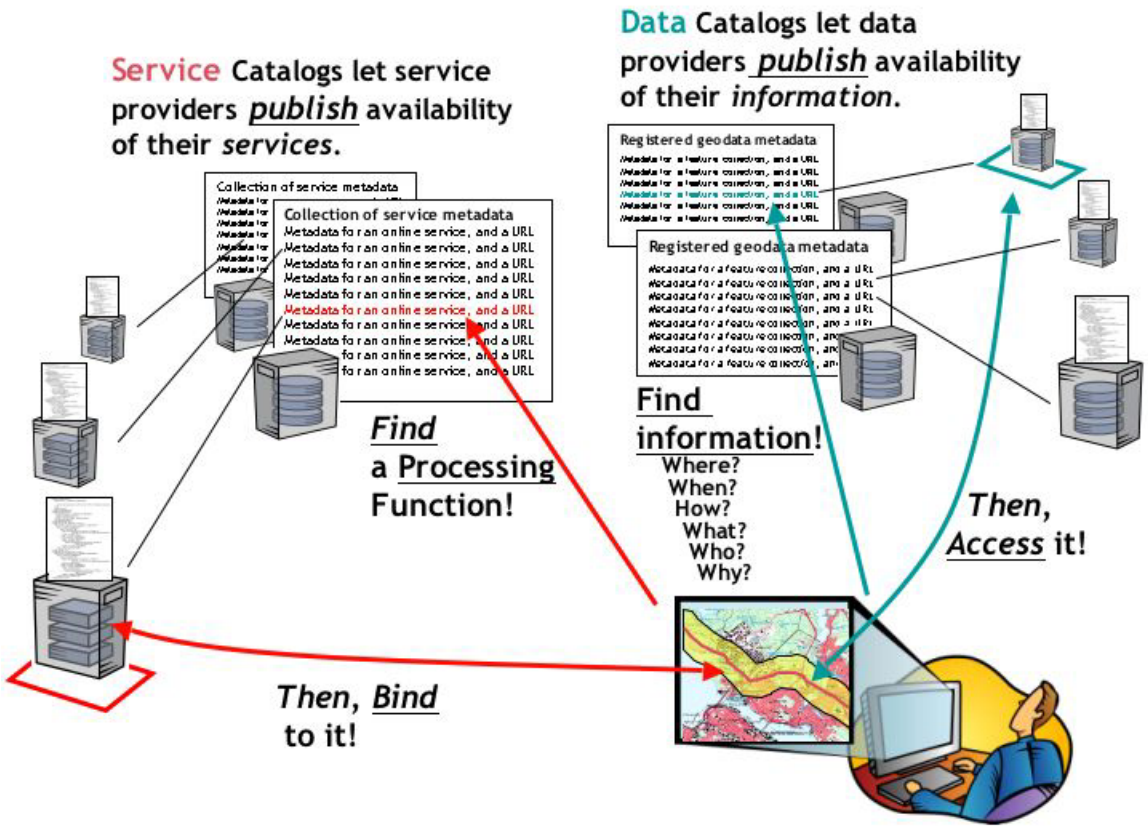
This framework also assumes a ***publish, find, bind*** pattern, which involves three roles:

- Service provider: publishes services to a broker (registry) and delivers services to service requestors.
- Service requestor: performs service discovery operations on the service broker to find the service providers it needs and then accesses service providers for provision of the desired service.
- Discovery service: helps service providers and service requestors find each other by acting as a registry or clearinghouse of services.

Services perform three kinds of operations:

- Publish: performed to advertise data and services to a discovery service. A service provider contacts the discovery service to publish (or unpublish) metadata describing a service's capabilities and network address.
- Find: performed by service requestors to locate specific service types or instances. Service requestors describe (to the discovery service) the kinds of services they’re looking for, and the discovery service then responds by delivering the results that match the request.
- Bind: accomplished when a service requestor successfully accesses and invokes services of the provider.

For example, an enterprise or a service vendor might host on a Web site a service for coordinate transformation, to convert the coordinate system of one map to the coordinate system of another map so the two maps can be overlaid and viewed together. A description of this service would be registered (published) in a service registry (a discovery service) so that a service requestor application could find it and "bind" to it.



OGC Web Services: A Framework of Capability

The OGC Web Services (OWS) effort focuses on defining, documenting, and testing interface specifications and information encodings that enable geoprocessing services and content to become seamless components of the emerging Web Services infrastructure. Software developers can use the OWS suite of OpenGIS Specifications to implement open interfaces that enable the following kinds of interoperability between their software and other vendors' software:

- Web Mapping with simple “picture” maps: manipulation and automatic registration and overlay of map images obtained from multiple online map servers and data repositories. (OpenGIS Web Map Service Implementation Specification).
- Discovery of cataloged data and service resources via browsing and query of metadata about the resources (OpenGIS Catalog Service Implementation Specification).
- Manipulation and query of distributed vector feature repositories (OpenGIS Web Feature Service Implementation Specification and Geography Markup Language (GML)).
- Manipulation and query of distributed coverage repositories. Roughly defined, "coverage" in OGC refers to image data – satellite imagery, digital elevation models, digital orthophotos, etc. (OpenGIS Web Coverage Service Implementation Specification).
- Web-based discovery of, access to, geolocation of and control of on-line sensors, including imaging devices (OGC discussion papers -- not yet approved specifications -- for Sensor Collection Service, Sensor Markup Language and Observations and Measurements).
- Semi-automated integration and automated use of data sets having dissimilar feature naming schemas and metadata schemas (GML and OpenGIS® Web Feature Service Implementation

Specification used with W3C technologies such as eXtensible Stylesheet Language: Transformations (XSLT) and XQuery. The use of these W3C technologies is still in the OGC discussion paper stage.)

- Communication of location, time, route, types of service, etc. across technology platforms, application domains, classes of products, and national regions in support of Location Services such as determining a route to a point of interest or determining where a traveler is located (OpenGIS Location Service Implementation Specification).

Fourteen OpenGIS Specifications have been completed and adopted by OGC and implemented by numerous vendors in their software products. See "Compliant or Implementing Products" at http://www.opengis.org. Others are still being developed in OGC's test beds, pilot projects and Technical Committee working groups.[1]

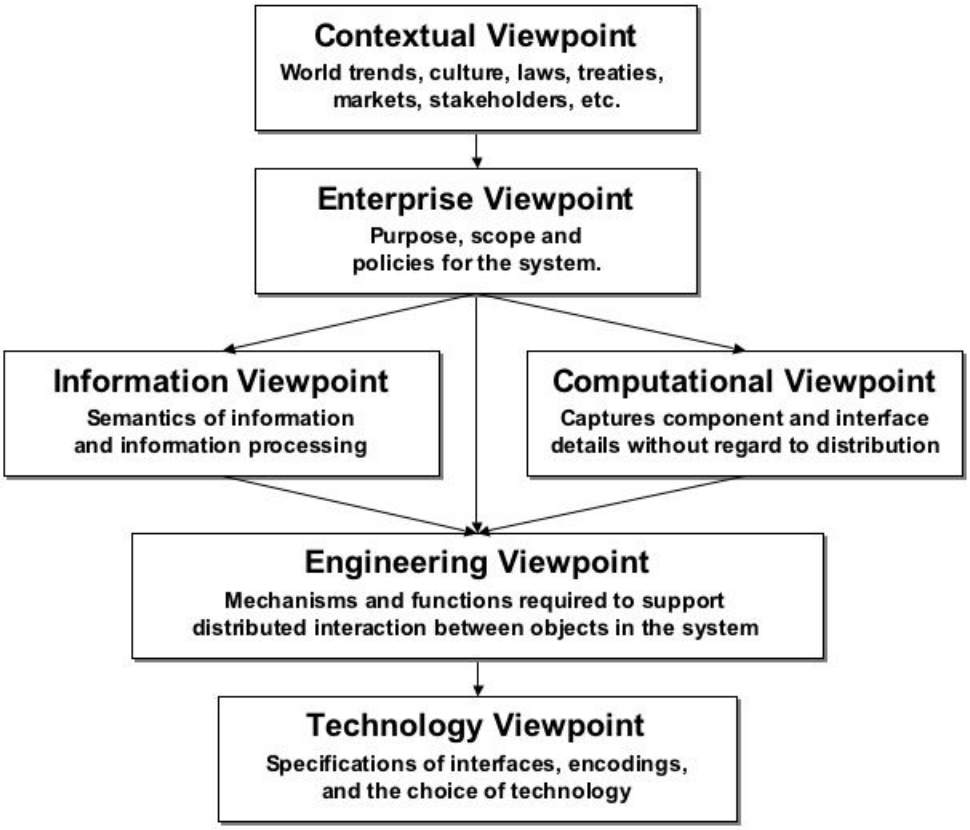
The Tools

Reference Model for Open Distributed Processing (RM-ODP)

The Web's distributed processing model of proprietary black boxes, open interfaces and registered resources demands that system planners apply a new kind of discipline in their work. One result has been the Reference Model for Open Distributed Processing (RM-ODP)[2], an international standard for architecting open, distributed processing systems. It provides an overall conceptual framework for building distributed systems in an incremental manner. The RM-ODP provides: 1) a way of thinking about architectural issues, and 2) a set of guiding concepts and terminology.

The RM-ODP defines standard concepts and terminology for open, distributed processing. In a generic way, the model identifies the top priorities for architectural specifications and provides a minimal set of requirements—plus an object model—to ensure system integrity. Five standard viewpoints are defined, which address different aspects of the system and enable the 'separation of concerns.' Often a sixth viewpoint, the Contextual Viewpoint is also considered. (See Figure below.)

[1] Portions of this section appeared in the January 2004 issue of the FGDC News, a publication of the U.S. Federal Geographic Data Committee.
[2] http://www.community-ml.org/RM-ODP/. Based on ISO/IEC 10746: Open Distributed Processing - Reference Model.



Before looking at how we use the RM-ODP as a tool, we look at one other tool, Model Driven Architecture (MDA).

Model Driven Architecture for Platform Independence

Model Driven Architecture (MDA) is an approach for designing and developing complex systems leveraging the capabilities of UML (Unified Modeling Language) models to manage complexity. In the pure form of MDA, the UML models themselves are sufficient to build a system. A system is initially designed using UML models to capture operational, computational, and information design decisions. Software tools then convert the UML models into executable code suitable for the target processing environment.

In reality, deriving executable code directly from UML is rarely possible. For most systems, the value of MDA is that it enables the architect to formally define the conceptual design for a system while leaving developers the freedom to implement that design using the technology appropriate for their requirements. Since the implementations are built from a common UML base, mediation services[3] can integrate the implementations into the greater enterprise.

Most of OGC's Implementation Specifications are for the Web platform. However, an OpenGIS candidate standard Application Objects Implementation Specification (AOS) is now being considered by the OGC membership for adoption as an OGC standard. AOS defines a set of vendor-neutral, Object-Oriented geometric and geographic object abstractions. It provides both an abstract object specification (in UML) and a JAVA specific profile to that specification. The language-specific binding specifications themselves serve as open Application Programmer Interface (API) specifications for these Application Objects.[4]

[3] A mediation service provides mechanisms for "homogenizing," or resolving the data mismatches and process sequence mismatches of heterogeneous Web services.
[4] "GO-1 Application Objects Discussion Paper," OpenGIS Project Document Number 03-064r2, Version: 0.3.0, Editor: Eric Bertel, Polexis, Inc.

The Geographic Objects Initiative provides an MDA foundation for service-based geoprocessing. There are two reasons for doing this: First, some organizations already have, or plan to build, service architectures that are different from the standard Web services architecture, and these organizations want to take advantage of commercial products that comply with OpenGIS Specifications. Second, organizations making major investments in spatial IT want to be sure that they will have as little trouble as possible adapting as the Web services architecture evolves or as the enterprise changes directions in its choice of technologies such J2EE or REST (see "Application of the Tools" below.).

Application of the Tools

The *RM-ODP Enterprise view* is where business processes are captured. We start by modeling a set of business processes. These processes are composed of information flows and the actions (processing) that are exercised on those flows. Each action can be decomposed into its atomic components. Primitive use-cases are used to capture these atomic actions. These use-cases comprise the set of action primitives from which all processes can be built. Information needs, like actions, are identified in the business process models and can be decomposed into primitive information types. These type definitions comprise the set of information primitives from which all information flows can be built.

Once completed, we have a language consisting of nouns (primitive information types) and verbs (primitive use-cases) that are sufficient to express all known business processes. The next question is – what can we do with this language?

To expand on the verbs, the *RM-ODP computational view* is used. Each primitive use-case identifies an information processing capability that the system must support. Each one of these capabilities corresponds to an information processing service. The behavior of that service is defined by its associated use case. Additional detail is provided to identify the interfaces, operations, and parameters needed to implement that behavior. This provides us with sufficient information to create UML models of the services.

To expand on the nouns, the *RM-ODP information view* is used. Each primitive use-case identifies the information exchanged as part of each transaction. UML models of the information types are developed to capture the specific data elements necessary to convey the required information. Where possible, existing information standards should be leveraged.

This set of Enterprise, Computational, and Information views comprise the *abstract framework*. This framework establishes a coherent picture of the range of business processes that can be performed as well as the computational and information resources needed to support those processes. Additional business processes, computational, and information resources can be added to these models as needed.

The next step is to understand the technical constraints that apply to the system. These constraints include available hardware and communications resources, operating systems, and perhaps most important, the Distributed Computing Platform (DCP). In order for two components to interoperate, there must be a set of standard protocols, encodings, and supporting services to enable that interaction. This collection of protocols, encodings, and services constitute a Distributed Computing Platform (DCP). Different DCPs have different strengths and weaknesses. The Java 2 Enterprise Edition (J2EE), for example, is a robust distributed computing platform that is suitable for high reliability and high performance applications. At the other end of the spectrum, REST is a minimalist approach for exchanging request/response messages. It is ideal for situations where middleware vendor and version incompatibilities can be expected. System designers examine the mission objectives of the system and select the DCP(s) that are appropriate. This collection of constrains is documented in the *RM-ODP Engineering* view.

The final step is to generate implementations of the services and information models defined in the computational and information views. Starting with the UML models, scripts and tools specific for the target environment (as defined in the Engineering view) are employed to generate interface code. While MDA techniques may not provide a final implementation, the code generated goes a long way to reaching that final set and provides assurance that the interfaces will be consistent with the abstract views.

Verification

MDA validates the models. Code generated from the MDA exercise will need some human modification. Identification of areas where MDA did not work properly can expose errors in the models themselves. Therefore the models and their implementation are mutually verifying.

Lifecycle maintenance requires traceability of business processes, from abstract to implementation. This clarity helps us to insert modifications to the environment in any view.

The Recipe [Suggestion: make this a sidebar.]

1. Identify the atomic actions that make up all known business processes.
2. Develop UML use cases for those atomic actions.
3. Define a service for each use case.
4. Develop a UML model for each service including interfaces, operations, and parameters.
5. Identify the information needs from each use case.
6. Develop UML models to capture a data schema for each information item.
7. Verify that the use cases and models are consistent and complete.

For each service or collection of services to be deployed:

8. Identify the performance and reliability needs.
9. Assess the capabilities of the computers and communications infrastructure.
10. Select a DCP that will provide sufficient reliability, throughput, and responsiveness.
11. Document the analysis and decisions (Engineering view).
12. Generate interfaces for this DCP from the UML models, or re-use existing interface specifications.
13. Make hand corrections as necessary.
14. Verify that the interfaces are consistent with the UML models.
15. Develop service code or wrap an existing service in the interfaces.
16. Test and deploy.

Conclusions

The RM-ODP provides important guidance to information system architects who are designing service architectures for enterprise information systems. MDA provides an additional layer of abstraction that enables an architect to define in a formal manner the conceptual design for a system while leaving developers the freedom to implement that design using the technology appropriate for their specific requirements.

The existence of a comprehensive open services framework (OGC's ORM) in the geoprocessing industry enables system architects to detail spatial information requirements in the Enterprise View with confidence that the requirements can be met by standards-based commercial off-the-shelf (SCOTS) component products. These component types are specified by naming OpenGIS Implementation Specifications in the Computational Viewpoint. The architect specifies in the Information View the data models (for which standards may also be available) that are also necessary to meet these information requirements.

References

For more information about the Reference Model for Open Distributed Processing (RM-ODP), See http://www.community-ml.org/RM-ODP/.

For more information about the Object Management Group (OMG), see http://www.omg.org.

For more information about OGC, see http://www.opengis.org. See OGC's "Spatial Web White Paper" and another white paper, "The Importance of Going Open," at http://www.opengis.org/press/?page=papers.