Chapter 2: Technology Frameworks for Information Sharing

Information-rich analysis efforts are characterized by their struggles with data preparation. This process can take months or years to complete (Waddell 2004), creating a situation where the "dirty little secret" of information analysis is that the majority of the time and effort is spent in data acquisition and formatting. The planning profession has generally ignored this problem, considering it a software issue which will improve with time and progress in the general field of information systems. This point of view seems reasonable, but much evidence suggests otherwise. If that is the case, it would seem that we would have observed significant improvements over the last few decades, but the results are mixed. We are digitizing less data, and using more data in our analyses, yet we continue to duplicate data development efforts, and we rarely implement systems whose data stays relevant from year to year. The problem is exacerbated by the fact that the organizations information moves between have different professional cultures, goals, and skills. Administrative divisions like property assessing have little in common culturally with the planning department, or a zoning board, or a local watershed protection group. These communities require their own methodologies for information processing, visualization and dissemination, and any proposal for improving information integration must not put restrictions on any organization's natural operational processes. A well-known concept in decision support is the idea that our systems should help people engage in the transformation of data into information into knowledge. Our current technologies have been good at providing decision support to individuals or small groups using self-contained systems, but when the system is like most planning analyses, having multiple, heterogeneous participants in every area—from the creation of data, to the modeling, to the presentation of results—they break down under the operational costs of the information transactions.

This situation suggests that the root causes of our data dilemma are not in what information systems or data converters we happen to use, but in defining an overall ***framework*** for processing information. A framework is an extensible structure for describing a set of concepts, methods, technologies, and cultural changes necessary for a complete product design and manufacturing process (CERN 2004). It is more than a set of software recommendations, or even a new technology proposal, but all those things in conjunction with the cultural and institutional changes necessary to effect real progress. This chapter presents a technology framework in which we can reduce costs, while developing urban information systems that hold up to increasing demands from participants in data input (data), information development (modeling), and knowledge creation (visualization and public participation). First, the concept of a planning support system is positioned generically as a distributed computing environment. This allows planners to leverage the systems that computer scientists have created for distributed information processing instead of inventing our own technology baseline. While there are a few alternative technologies for doing distributed computing, a Web Services framework is chosen. This decision helps solve the next issue, which is to develop planning-specific decision support systems within the distributed computing environment. In a Web Services framework, domain-specific information models are developed in a semantic meta-language like RDF or XML. While these tools have various pros and cons, Web Services software available today is designed to use XML, and the practicality of using RDF has yet to be shown. In the following chapters, we adopt the Web services framework, and use it to prototype a new urban information system based on data and analysis services. This is presented through a series of use cases relating to data publishing, urban modeling, and participatory GIS where case-specific solutions are developed. Finally, a full system is presented in Chapter 7, and the MassGIS buildout analysis is presented in this new framework. The XML vocabulary is called ***Planning Analysis and Modeling Markup Language***, or PAMML, and the Web Services built on it are referred to as ***PAMML services***.

An introduction to distributed computing

A distributed computing environment is one in which information and the applications that make use of it are physically located on different computers. In order for these computers to know that others of their kind exist, and how to talk to them, computers need a whole host of hardware and software. For the purposes of this work, we will assume that communication occurs via what is commonly called the Internet, which includes Ethernet and TCP/IP.

Figure 2-2:
Abstract 3-Tier Architecture

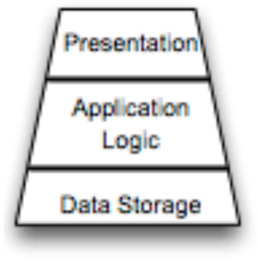
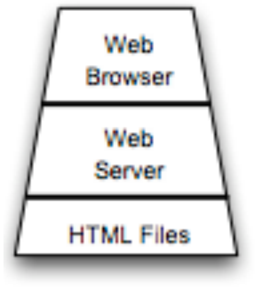


Figure 2-2:
Web 3-Tier Architecture



In this environment, an information warehouse is called a ***resource***, and the system that provides information is generally called a ***service***. So in this parlance, information, or data, is retrieved from a ***resource*** through interaction with a ***service***. The agent that requests information—for example a person, a computer or a computer program—is called a ***client***. What has just been described is usually called a "three-tier architecture" in computing. This architecture underlies most of the important systems in use today, including e-mail, instant messaging, and the World Wide Web.

In this architecture, any information store, such as a parcel database or an address book, becomes an abstract concept. The actual data can only be accessed by making a request to a service, which serves as the gatekeeper to the data. PAMML is a language that describes how to build services, so that different services can be expected to reliably interact with one another.

This architecture is quite complex and difficult to implement in practice, so why bother? The best answer is that distributed computing is flexible enough to mirror the organizational situations we encounter in the real world. For example, if everyone was required to have an email server on his or her computer and they could only read their email on that computer, it is doubtful that email would be in widespread use today. In government, our interest centers on the distributed nature of information and domain knowledge. For example, the assessing department uses parcel data more than any other agency. Therefore, they are best able to make sure that parcel information is up to date and captures the knowledge about parcels required for municipal administration. The same applies to other domain experts, such as traffic engineers, natural resource managers, and infrastructure providers. Unlike most of these other organizations, planning practice is defined by the ability to integrate and analyze information from other domains. If successful planning outcomes were not so dependent upon having access to the right information, such close attention would not have to be paid to the information infrastructure of all the professions involved in collection information about places.

The IT world offers various solutions for implementing distributed computing applications. EDI, or electronic data interchange, is decades old and has been favored by organizations with high security and reliability needs like banks and airlines. While the technology is proven, participation in an EDI system requires a great deal of programming and system administration skills, which would eliminate the potential participation of most local governments and non-profits.

In the early 1990s a system called CORBA became popular. Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network. CORBA has been widely used to connect corporate information systems, and is getting some attention in the GIS field (Preston, Clayton and Wells 2003). A full analysis of this is beyond the scope of this paper, but in general, CORBA seems to be too "tightly coupled", requiring too high a level of coordination and cooperation between agencies, despite its language and operating system independence (Gottschalk 2000).

Web Services

As personal computing and the World Wide Web gained popularity in the 1990s, the IT landscape changed. Information sharing and processing was no longer the sole purview of big corporations. There was suddenly a vision of all organizations and individuals participating in a global information community. The old systems were not offering answers to these new challenges, so computer scientists looked at the Web and tried to understand why it had been so successful. It was found that the Web architecture requires only a minimal set of standards—HTTP as the basic application level protocol, and HTML for formatting information—but it delivers the ability to communicate without centralized planning or control, and to integrate a heterogeneous mix of platforms and programming models (Curbera 2001). The result is a very shallow interaction model between a very heterogeneous set of clients and servers that allows simple things, like sending a text file to someone's computer, to be easy; and complicated things, like buying a book with a credit card, to be possible.

The Web still has many limitations. HTML was designed as a way to mark up text for display, and HTTP is best at handling communications between only two computers at a time. In order to improve upon the quality of information available on the Web, and the systems that enable multi-computer, multi-organization transactions, something more was needed. XML, the successor to HTML, and Web Services, a descendant of EDI and CORBA built on Web standards, address these needs.

XML and XML Schema defined

XML stands for eXtensible Markup Language. It is a meta-language—a language designed for developing other languages. XML was developed as a way to tag information with metadata and enforce structural rules without requiring that the information be stored in or adhere to the strict rules of a database. It has proved to be a highly successful strategy, as the language is barely five years old and is already extensively used to formally describing information that does not fit nicely into the relational database paradigm. What XML provides is a consistent structure and a way of formally describing a language's vocabulary. The World Wide Web Consortium defines XML's design goals as follows (World Wide Web Consortium 2004):

1. XML shall be straightforwardly usable over the Internet.

2. XML shall support a wide variety of applications.

3. XML shall be compatible with SGML.

4. It shall be easy to write programs which process XML documents.

5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.

6. XML documents should be human-legible and reasonably clear.

7. The XML design should be prepared quickly.

8. The design of XML shall be formal and concise.

9. XML documents shall be easy to create.

10. Terseness in XML markup is of minimal importance.

The benefit to writing a language in XML is that you can take advantage of a vast collection of software already developed to process XML, and only write the software that deals with the specifics of your particular language. Furthermore, one XML language can use others to describe generic entities. For example, XML language developers do not have to describe how a person's address should be written. They can simply use an XML address language developed by another information community (such as software companies that develop address book software). More importantly, a great deal of infrastructure needed to make an application work is common to all applications, such as security, authentication, field validation, etc. Using XML makes it possible for a language writer to be confident that their language can take advantage of advances in these areas without requiring major changes to their own work.

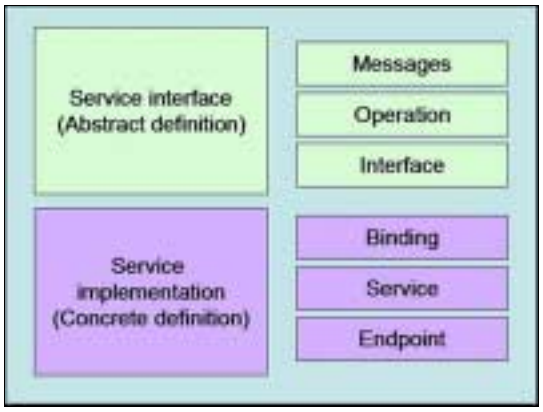
The way one develops an XML-based language is to write a rulebook. This is done in an XML language called XML Schema. This document functions as a dictionary—defining the set of terms that can be used—and also as a grammatical reference—enforcing rules about how words are put together to make sense. Additionally, XML Schema has the ability to reference other XML Schemas. This makes it possible to leverage existing work in related areas. PAMML can use this mechanism to avoid reinventing the wheel in the areas of networking, identity management, databases, and GIS. For example, whenever a PAMML document needs to reference to a resource located somewhere on the Internet, the World Wide Web Consortium's (W3C) XLink vocabulary can be used to identify the resource. Database access may take advantage of W3C's evolving XQuery vocabulary. In the geographic information systems field, a number of OpenGIS Consortium (OGC) specifications will be used. GML (Geography Markup Language) will be a supported data set format, and GML will also be used as the "native" geographic object language. WFS (Web Feature Service) will be a supported data format, in concert with the Filter encoding specification, which defines queries on geographic data.

Web services defined

"Web services" is an umbrella term used to describe systems that allow computer software to communicate using XML as a messaging language. The different communication implementation strategies go by many names (the most well known being SOAP, or Simple Object Access Protocol). However, the implementation strategies are not important in this context. What is most important is that all Web services strategies use the well-known and widely implemented Internet protocol for communication—HTTP—the foundation upon which all Web sites operate. While HTTP's simplicity has many drawbacks, the advantages are numerous. The most obvious is that most organizations already have a Web infrastructure in place, so the most basic Web Services implementations can be handled in a familiar way, and the extensive range of Web software can be used to develop and run new Web Service-based applications. The other important aspect of Web services is that they use XML for passing messages between computers, preserving the transparency that has made XML so popular and useful.

The description of a Web service can be modeled in two parts. In the abstract part, WSDL describes a Web service in terms of messages it sends and receives through a type system, typically W3C XML Schema. Message exchange patterns define the sequence and cardinality of messages. An ***operation*** associates message exchange patterns with one or more messages. An ***interface*** groups these operations in a transport and wire independent manner. In the concrete part of the description, ***bindings*** specify the transport and wire format for interfaces. A service ***endpoint*** associates network address with a binding. Finally, a service groups the endpoints that implement a common interface. Figure 2-3 shows the conceptual WSDL component model.

Figure 2-3: WSDL conceptual model


Service interface (Abstract definition)
Messages
Operation
Interface
Service implementation (Concrete definition)
Binding
Service
Endpoint

Some alternative frameworks

As mentioned earlier, precursors to Web services were EDI and CORBA. Also in this group are other frameworks having their roots in computer programming languages, like RMI (remote method invocation) and DCOM (distributed component object model), and programming languages in general. The problem with these systems is that they are too "tightly coupled," meaning that the two organizations wanting to exchange information with each other need to know a great deal about the other's systems and use similar technologies to build the communication software. When one organization changes their database or a piece of code, it is likely that the other organization will have to do the same. This type of system will only work out if there are a limited number of groups involved and they have a strong motivation to collaborate.

Systems that seek to integrate organizations on a larger scale need "loosely coupled" frameworks. In a loosely coupled system, most aspects of an organization's information system are hidden, or abstracted, from the world. There is no need for particulars such as operating system, database software, and even the information model, to be shared with others. Organizations exchange information via computer-to-computer messages, which are understood by all the partners in the exchange. The earlier description of XML and Web services obviously fits this description, but two other frameworks seek to do similar things, UML and the Semantic Web.

UML

"The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components" (Object Management Group 2003, page xxv). This notion of a standard way to write a system's blueprints makes UML a candidate for developing a generic planning information system, because this helps to fulfill the requirements of a loosely coupled system. Its strengths are that its primary output is a visual diagram; it can be used to describe a system in a very loose, unspecific manner; but can also be highly specific if necessary, retaining the features of a formal method. As stated by Muller, "A method defines a reproducible path for obtaining reliable results. All knowledge-based activities use methods that vary in sophistication and formality. Cooks talk about recipes…architects use blueprints, and musicians follow rules of composition. Similarly, a software development method describes how to model and build software systems (Muller 2000)." The UML method represents the software industry's consensus on how to graphically describe a software system.

The UML's strengths are also its weaknesses. While a graphic notation is great for humans, it is not computer readable. Also, generalized UML models are ***too loose***. It is difficult to ensure that different applications can interpret the model in the same way and therefore interoperate. Software engineers use the UML to explain high-level ideas about system design, not to directly specify system execution. There have been efforts to overcome these limitations by specifying an XML vocabulary for UML diagrams, and develop standards for highly specific models, but these efforts quickly begin to look like Web services, and will probably end up as such.

Web Ontology Language

The Web Ontology Language (OWL) is a relatively new initiative from the World Wide Web Consortium. It represents a major step in the maturation process of efforts to define formal semantics about Internet-accessible information content. These efforts began with a DARPA-funded effort called DAMML+OIL and more recently has moved forward under the Resource Description Framework (RDF) specification (http://www.w3.org/TR/rdf-primer/). OWL and RDF are part of a broad effort geared towards improving the description of information on the Web, called the Semantic Web. The World Wide Web Consortium (W3C) defines the Semantic Web as, “the representation of data on the World Wide Web…It is based on the Resource Description Framework (RDF), which integrates a variety of applications using XML for syntax and URIs for naming” (*http://www.w3.org/2001/sw/*). Here is the W3C’s definition of OWL:

“OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called an ontology. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S, and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web. OWL is a revision of the DAML+OIL web ontology language incorporating lessons learned from the design and application of DAML+OIL (*http://www.w3.org/TR/owl-features/*)."

OWL and RDF have many similarities to XML Schema. In fact, they both use XML Schema as their recommended expression language. The major difference between XML Schema and the semantic languages seems to be in the amount of flexibility allowed in defining relationships. XML Schema is limited in its ability to say that one thing is like another without defining them as being of the same data type. It is also difficult to construct relationships between resources without prior cooperation between the developers of those resources. On the other hand, OWL and RDF have very specific language constructs to explicitly define the relationships between objects. This makes the semantic languages very good at creating taxonomies and reconciling the different taxonomies that various organizations may create. Where the semantic languages run into trouble, however, is when one tries to build a data-centric application. The very flexibility that is such a positive feature in some situations becomes a negative when an application must count on a certain data field being present in every object it encounters (Forsberg and Dannstedt 2000).

OWL may eventually become an appropriate framework in which to build a collaborative planning support system vocabulary, but the technology is too young to consider for practical experimentation at this time, and this project did not identify any information modeling issues that were beyond the capabilities of XML Schema to handle.