Emergence of Distributed Engineering Web Services

Jun Peng[1], David Liu[2], Jinxing Cheng[3], Charles S. Han[4] and Kincho H. Law[5]

[1] Research Associate, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305; PH 650-725-1886; junpeng@stanford.edu
[2] Ph.D. Student, Department of Electrical Engineering, Stanford University, Stanford, CA 94305; davidliu@stanford.edu
[3] Ph.D. Student, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305; cjx@stanford.edu
[4] Consulting Assistant Professor, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305; chuck.han@stanford.edu
[5] Professor, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305; PH 650-725-3154; law@stanford.edu

Abstract

This paper presents the basic concepts of the web services technology and its potential applications in Civil Engineering. There are various communication protocols that are now available for building distributed engineering web services. This paper presents three example applications utilizing some of these protocols. These examples are employed to demonstrate that the web service approach is a promising paradigm for integrating large engineering software applications.

Introduction

The world of web services is evolving very quickly. Simply defined, a web service is a combination of applications and data that can be accessed from any web-enabled devices. Using a set of standardized protocols, web services can be universally accessed and deployed, independent of the underlying operating environment. The basic idea behind web services is to adopt the web programming model for developing generic applications that are not necessarily browser-based. The goal is to provide a platform for building distributed applications that utilize software components running on different operating systems, computer platforms and devices. Heterogeneity is assumed for the software components, which can be built using different programming languages, operating systems, hardware platforms, and vendors. Acting as adapters for complicated process components and legacy applications, web services allow disparate systems to work together.

The web services model is also becoming a favorite approach for integrating engineering applications in that it not only improves the flexibility and efficiency of engineering simulations but also makes the composition of engineering services easier. A typical engineering simulation may involve a number of software applications that run on geographically distributed computers. For example, the architects, structural engineers, and construction team of a project may reside in different locations and use separate computer systems and software packages for engineering analysis and design. The simplicity of the web services model makes it possible to build a

complicated software system incrementally. The feature of incremental improvement is very important for engineering software applications, which are typically constantly evolving.

Emergence of Web Services

As software becomes ever more complex, there is a shift from coding as the focus of programming to a focus on integration, as illustrated in Figure 1. Traditionally, large programs are partitioned into subtasks of manageable sizes. The subtasks are assigned to programmers who code the instructions in a programming language. The resulting program segments are subsequently submitted for integration. As more and more program segments are pre-constructed and packaged, larger portion of the overall software engineering effort needs to be spent on integration.

Software Integration. Software integration takes place in many forms. The early attempts are based on code reuse. The simplest method is to copy the source code to wherever the desired functionality is needed. There are significant drawbacks to this approach, ranging from compiler incompatibility to difficulty in maintaining duplicate copies of code. To deal with these drawbacks, shared libraries are used in place of copied code. Software components written in a programming language are compiled into shared libraries. The shared libraries have public interfaces, through which the users can invoke the functions contained in the libraries. Generally speaking, software integration based on code reuse assumes that the ownership of the reused software components belongs to the users of the software components. In addition, the software components are executed on the same machine as the invoker (client) of the components.

The development of network computing, especially the emergence of the Internet, allows software components to be distributed to multiple machines. Each software component runs as a separate process, communicating with each other by exchanging messages. This software integration model is called distributed component model. As the distributed component model evolves, software components are required to have well-defined interfaces and constraints, and are normally managed in a decentralized manner. The distributed components whose interfaces are published on the web are generally referred to as web services (Roy and Ramanujan 2001).

Web services can be regarded as the atomic building blocks for service integration. The functionalities provided by the web services are composed together to form an integrated service. Although the web services are distributed and heterogeneous, they are utilized as if they were locally available to the megaservice. An infrastructure is needed to provide the support for the composition of the service functionalities. Communication messages are exchanged among the web services to coordinate the execution of the web services and to exchange data among the web services. To achieve interoperability, a set of communication protocols is needed for exchanging data among the web services.

Communication Protocols. Distributed components need communication protocols to exchange messages, so that each of these distributed components can interoperate as a unified system. There has been an emergence of standardized communication protocols that can be used to build web services. Distributed components may be written in different languages, and can be compiled by different compilers, while they communicate with each other via standardized protocols. Example communication protocols are Common Object Request Broker Architecture

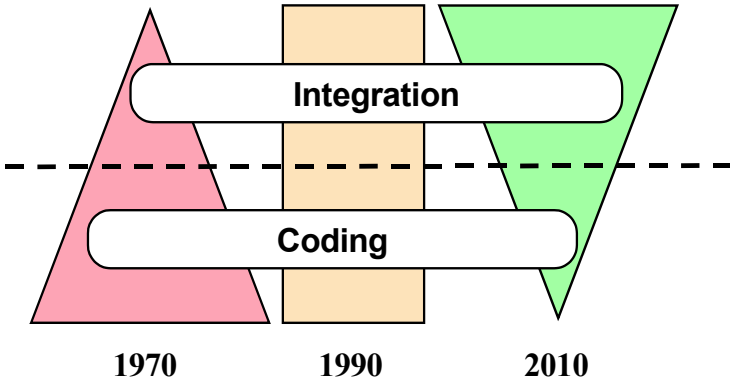

Figure 1. The Trend of Software Development (from Wiederhold 1996)

(CORBA) (Pope 1998), Microsoft's Distributed Component Object Model (DCOM) (Eddon and Eddon 1998), Java Remote Method Invocation (RMI) (Pitt and McNiff 2001), and Simple Object Access Protocol (SOAP) (Box et al. 2000).

The Common Object Request Broker Architecture (CORBA) is a source interface standard being promoted by the Object Management Group (OMG), an industry standard consortium. Everything in the CORBA architecture depends on an Object Request Broker (ORB). The ORB acts as a central object registry where each CORBA object interacts transparently with other CORBA objects located either locally or remotely. CORBA relies on a protocol called the Internet Inter-ORB Protocol (IIOP) for remote objects. Each CORBA server object has an interface and exposes a set of methods. To request a service, a CORBA client acquires an object reference to a CORBA server object. The client can make method calls on the object reference as if the CORBA server object resides in the client's address space. The ORB is responsible for finding the CORBA object's implementation, preparing it to receive requests, communicating request to it, and carrying the reply back to the client. A CORBA object interacts with the ORB either through the ORB interface or through an object adapter. Since CORBA is just a specification, it can be used on diverse operating system platforms as long as there is an ORB implementation for that platform.

The Microsoft DCOM, extended from Component Object Model (COM) and more recently in COM+, provides a distributed object framework as an extension of the OLE (Object Linking and Embedding) facility. OLE allows objects to be linked by reference between types of documents and objects to be embedded in other objects. DCOM supports remote objects by running on a protocol called the Object Remote Procedure Call (ORPC). The ORPC layer is built on top of standard remote procedure call (RPC) and interacts with COM's runtime services. A DCOM server is a body of code that is capable of serving up objects of a particular type at runtime. Each DCOM server object can support multiple interfaces each representing a different behavior of the object.

Java RMI relies heavily on Java Object Serialization, which allows objects to be marshaled (or transmitted) as a byte stream. Each RMI server object defines an interface which can be used to access the server object outside of the current Java Virtual Machine (JVM) and on another machine's JVM. The interface exposes a set of methods that are indicative of the services offered by the server object. For a client to locate a server object for the first time, RMI depends on a registration and naming mechanism called an *RMIRegistry* that runs on the Server

machine and holds information about available server objects. A RMI client acquires an object reference to a JRMI server object by doing a lookup for a server object reference and invokes methods on the server object as if the RMI server object resides in the client's address space. RMI server objects are named using Uniform Resource Locator (URLs).

SOAP is an XML-based messaging protocol for information exchange in a decentralized, distributed environment. SOAP is essentially a flexible form of the traditional remote procedure call (RPC) mechanism for gluing heterogeneous distributed applications together. All the communication messages are encoded in XML (eXtensible Markup Language) format. XML-based messaging allows the applications running on different platforms to understand the exchange message without the need to conduct data marshalling. Another key advantage of SOAP is its simplicity, which enables its quick and wide adoption. SOAP is intended to provide the basic functionality as a messaging protocol for invoking web services.

With the proliferation of Internet-enabled services, there has been a push to allow such services to be integrated or composed to construct a *megaservice*. A centralized control and distributed data flow model, FICAS (the Flow-based Infrastructure for Composing Autonomous Services) has been proposed for service integration (Liu et al. 2002). FICAS is a collection of software modules that support the construction of web services, facilitate the specification of the megaservice, and enable the efficient execution of the megaservice. The distribution of data communications is enabled by a metamodel defined for web services, which separates the data communications from the control processing in the services. Web services conforming to the metamodel can be coordinated by a centralized controller, while data communications are distributed allowing *point-to-point* communication among the services. The distribution of computations is enabled by mobile classes, i.e., dynamic processing routines that can be loaded onto a web service to process data local to the service. By moving computations closer to data, the amount of data traffic can be significantly reduced. FICAS provides a test bed to examine the techniques as well as the performance impact of distributing data and computations for service composition.

Integration Models. Depending on how the software components are connected and managed to form the megaservice, the models to integrate software components can be roughly categorized into two groups, namely tightly coupled and loosely coupled. When software components are managed under a single administrative domain, we refer this type of integration model as tightly coupled component model. The software components follow a set of proprietary rules that allow access to software components cross the physical border of a single machine. For instance, software components are made to use a low-level platform-independent data format for representing data exchanged among distributed components over the network under CORBA (Pope 1998). For software components managed under multiple administrative domains, loosely coupled component model is preferred for integration. The software components in the loosely coupled model exist as autonomous services. The loosely coupled component model assumes that each service is controlled by its own service provider, i.e. the management of the autonomous services is hidden from their users. A typical web services application uses the loosely coupled component model. Unlike in the tightly coupled distributed object systems where all the pieces of an application are deployed at once, the web services model allows services to be added as needed. The connections to a new web service can be established during the system runtime with excellent interoperability, scalability, and manageability.

Web service applications range from comprehensive services such as storage management and customer relationship management to more specific services such as travel reservation, book purchasing, weather forecasts, financial data summaries, and newsgathering. Other services include engineering simulations, logistics, and business services. The objective of our research in web services is to develop methodologies that can effectively wrap legacy applications and make them accessible over the network. Many engineering applications require frequent communications among the services, where the tightly coupled data exchange needs to be supported. Another scenario involves large volumes of data exchange among the services, where data communications need to be distributed to improve the performance of the applications.

Application Examples

The following presents three example applications that we have developed to demonstrate the use of different communication protocols in building distributed engineering web services.

A Web Service Example for Building Design. The first example application is a distributed service architecture that allows building design web services to be incorporated into a modular network-enabled infrastructure (Han et al. 1999). CORBA is used as the standardized communication protocols. Figure 2 shows the conceptual Internet-enabled framework for a distributed framework with three web services and a broker. In this framework, each individual service adheres to a three-tiered architecture. The first tier, a communication protocol interface, gives the application services a common means to send and receive design data over the Internet. The middle tier, the common product model interface, is a standard protocol that describes the design data. The third tier is the core of the design service – the design service extracts the appropriate information of the building design through the common product model interface and either modifies the design data or generates a report based on the analysis of the data. As shown in Figure 2, the broker does not need the product model interface that is present in the services. An application package will register with the broker to advertise its services in the infrastructure. Another service will query the broker for the existence of services in the distributed service architecture. The registration and query service is based on a predetermined constraint language, but the broker does not have to be aware of the underlying product model that is being used to exchange design data between services.

For the building design web services, the communication protocol layer is implemented using CORBA. Legacy applications are wrapped as new web services and integrated into the distributed service architecture. In this example, the legacy application is the disabled access code analysis service, which can be used to automatically check if a building design conforms to the disabled access code. This legacy application shown in Figure 3 has its own proprietary communication protocol interface and a different product model interface. Therefore, the key of the service integration is to map the common product model to the legacy application's product model. Once the design data is mapped, the service launches the legacy application which then sends the stream of design data to the server where another application, code compliance checker, resides. The legacy server application receives the design data, runs a code compliance analysis on the building model, and generates a web page.

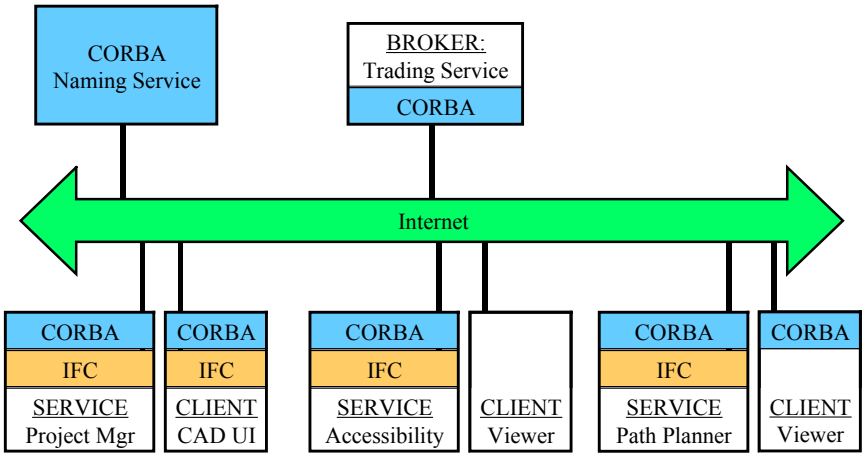

Figure 2. Web Service Architecture for Building Design

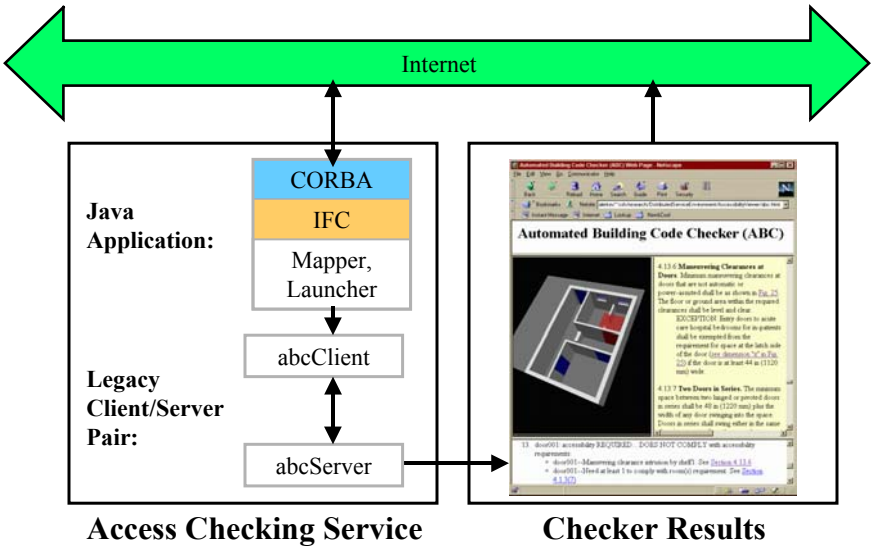

Figure 3. The accessibility analysis service and the generated report

A Web Service Example for Structural Analysis. The second example is a web service framework that facilitates the utilization and the collaborative development of a finite element structural analysis program (Peng and Law 2002). In the collaborative framework, a finite element analysis program is constructed as a web service to allow easy access to the analysis program and the analysis results by using a web-browser or other application programs, such as MATLAB. Although the finite element analysis program can be viewed as a single web service from the users' perspective, the analysis program can actually be running on disparate computers. The core service is built upon an object-oriented finite element program (Mckenna 1997) and it serves as the entry point for users' requests. Elements, materials, and solution strategies can be constructed as web services and run on distributed computers to facilitate structural analyses.

The overall system architecture of the Internet-enabled collaborative framework is schematically depicted in Figure 4. The architecture defines the dependency and the interaction among the participants. In this framework, the structural analysis core program is running on a central server as a web service. At the heart of the core web service is a set of interfaces that allows jobs to be submitted to the core server, the core service to run those jobs, and the results of the jobs to be returned to the client. Building a structural analysis program as a web service provides greater flexibility and extendibility than traditional structural analysis programs development, in which all the software components are typically bundled as a single package.

- In this collaborative framework, the users play the role of clients to the central finite element web service. The users can remotely access the core program through a web-based user interface or other application programs, such as MATLAB. The users can specify desirable features and methods (element types, efficient solution methods, and analysis strategies) that have been developed, tested, and contributed to the framework by other participants.
- A standard interface/wrapper is defined to help element developers to build finite elements as web services. The element routines can be written in languages such as Fortran, C, C++ and/or Java as long as they conform to the standard interface, which is a set of pre-defined protocols to bridge the element code with the central server. An element service needs to register its location and other related information to the core, and the element service can then be accessed remotely over the Internet. Treating element routines as a web service provides an effective way to wrap legacy element code. Since the element source code is not exposed to the core server, the collaborative framework also supports the building of proprietary element services.
- To support many web services participated in the system, the core server must be able to differentiate the services and locate appropriate services for specific tasks. One approach to solve this problem is to create a registration and naming service, where each participant could register its service to the core with a unique service name and address of the service. With the registration and naming service, the core can obtain the references to the web services it wishes to use.
- A COTS (commercial off-the-shelf) database system can also be linked with the central server to provide the persistent storage of selected analysis results (Peng et al. 2003). In this design, the database system simply serves as a web service to provide data storage and to facilitate data management. A customized interface to link the analysis core with the database service is implemented. The users can query the core server for useful analysis results, and the data retrieved from the database through the core server is returned to the users in a standard format.

In the prototype implementation, Java RMI is chosen to handle the network communication for distributed element services. The prototype system is employed to conduct a nonlinear dynamic analysis on an 18-story two-dimensional bay frame model. In the example model, the *ElasticBeamColumn* element is built as a web service. Figure 5 illustrates the interaction among the web services during a simulation of the model. The analysis core service is running on a server computer called *opensees.stanford.edu*. The developed ElasticBeamColumn element service is running on a computer named *galerkin.stanford.edu*. A MATLAB-based post-processing web service is deployed on *epic21.Stanford.edu*. Since it is the core server that handles the communication with the element service, clients only need to know the location of the central server (*opensees.Stanford.edu*) without the awareness of the underlying distributed model.

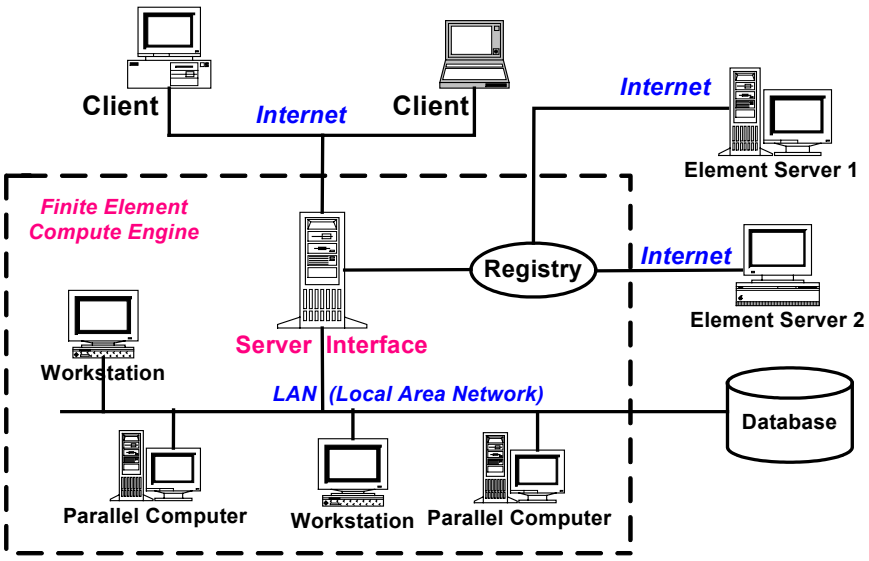

Figure 4. System Architecture for Structural Analysis Web Services

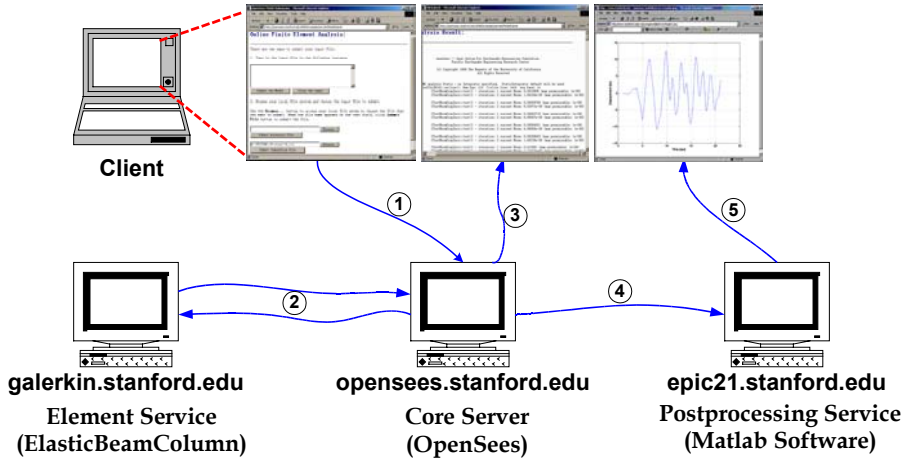

Figure 5. Interaction of Distributed Web Services

A Web Service Example for Project Management. The third example is a web services application for construction project scheduling. The FICAS model is employed to implement the web services framework for integrating a variety of project management applications, which involves large volume of engineering data communication (Liu et al. 2003). In this example, the web services are linked together through an integration framework to compose software applications and to form a megaservice. Proprietary software applications, such as Microsoft Project, Excel, Primavera Project Planner, Vite SimVision, and 4D Viewer are wrapped as web services that export their functionalities. Although the applications run on heterogeneous platforms and utilize different interfaces, they can be accessed homogeneously through standard web services interfaces. A project scheduling megaservice is constructed by composing the web services together (Cheng et al. 2003). Functionalities from various software applications are utilized to complete a specific engineering task. The prototype also incorporates a variety of

devices ranging from PDA, web browsers, desktop computers, and server computers. Using the infrastructure, construction personnel can conduct project management with the latest project information on the construction site, in a truly ubiquitous fashion (Liu et al. 2003). In short, by using the web services model to develop the integration framework, project management applications can collaborate regardless of location and platform.

The goals of the distributed integration infrastructure are to link software applications to act collaboratively on a project, and to allow access to the results at all times and locations. The distributed integration infrastructure employs the Process Specification Language (PSL) (ISO 2003) as the information interchange standard among different project management tools. PSL was initiated by the National Institute of Standards and Technology (NIST) and is emerging as a standard exchange language for process information. As shown in Figure 6, a communication server and a database system are used to serve as the backbone of the system. The communication server is responsible for listening requests from clients, which can be the application services or the client devices. When the server receives a request, it broadcasts the request to different communication agents. The communication agents then pick up the request and process it. In addition, an active mediator is built to act as an information broker between the client devices and the information sources. The user can send a request from a web browser (in an XML format) to the communication server and the database via the active mediator. The results retrieved from the database are then returned to the mediator, which filters and transforms the results suitable for display on the client device.

The network communication mechanism for the distributed web service infrastructure is illustrated in Figure 7. Java socket communication is used as the medium between the communication server and the agents. A communication agent includes an event listener, an event dispatcher, and a data mapper. The messages in the system include control messages and data messages. Control messages, such as invoking and terminating requests, are typically small in size. Data messages, such as the project scheduling information and organization information, on the other hand, are usually bigger in size. The event listener receives control messages, while the event dispatcher sends out control messages. The data mapper is responsible for sending and receiving data messages.

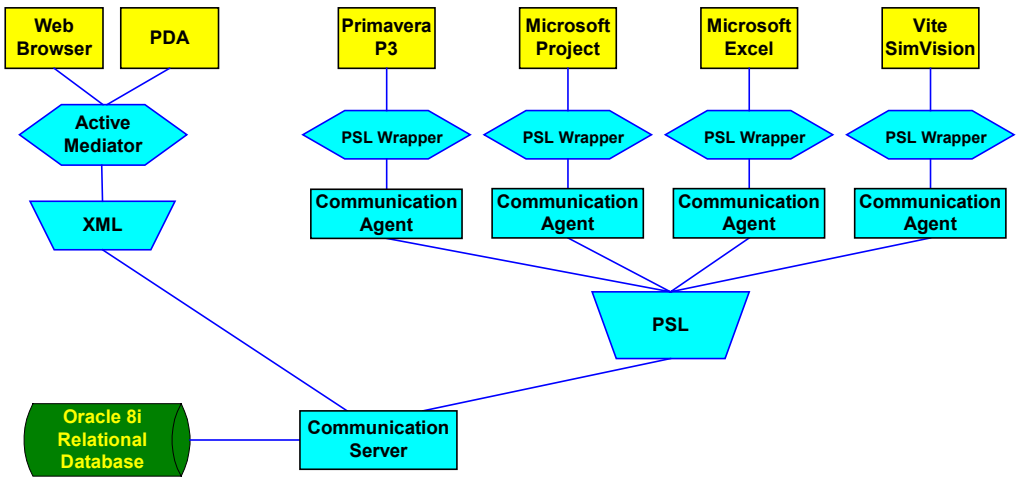

Figure 6. Web Service Infrastructure for Project Scheduling

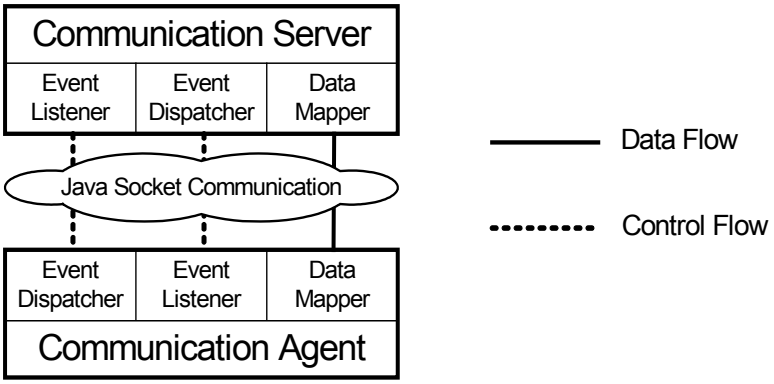

Figure 7. Network Communication Mechanism for the Web Services Infrastructure

Conclusions

This paper has presented three example applications that demonstrate the applicability and flexibility of the web services technology. Although the mechanisms and protocols that these applications employ to achieve distributed computing are different, the approach that each of them takes is more or less similar. Integrating distributed engineering applications as web services provides an effective mechanism to make legacy applications accessible to a broader group of users. Developers can collaborate to build sophisticated engineering applications by developing distributed modules that provide portions of the desired functionalities. In addition, by applying optimization techniques, such as the distribution of data communications (Liu et al. 2002), the web services approach becomes particularly suitable for large scale engineering simulations which typically generate and transfer large volume of data.

Web service is still an emerging technology and many improvements need to be made. First, many general-purpose features, such as security, reliable message delivery, and transactional semantics, are needed to facilitate the development of web services. Second, specifications for the web services need to be standardized. The lifecycle of the specifications typically progresses from proposal to *de facto* standard to actual standard. While researchers are continuing to propose new specifications, standardization is required for adoption. Third, toolkits and frameworks for developing web services need to be improved. To promote scalability in the integration of web services, programming models need to shift from procedural call style services toward specification-centric services. As these new developments continue, the web services technology will be applied more widely for developing engineering software applications.

Acknowledgements

This work has been supported in part by the Pacific Earthquake Engineering Research Center through the National Science Foundation under Award number EEC-9701568, National Science Foundation Grant Number CMS-0084530, the Center for Integrated Facility Engineering at Stanford University, the Product Engineering Program at National Institute of Standards and Technology, and an equipment grant from Intel Corporation.

Reference

Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. (2000). *Simple Object Access Protocol (SOAP) 1.1*, W3C Note, http://www.w3.org/TR/SOAP.

Cheng, J., Law, K. H., and Kumar, B. (2003). "Integrating Project Management Applications as Web Services," submitted to *the 2nd International Conference on Innovation in Architecture, Engineering and Construction*, Loughborough University, UK.

Eddon, G., and Eddon, H. (1998). *Inside Distributed COM*, 1st Ed., Microsoft Press, Redmond, WA.

Han, C. S., Kunz, J. C., and Law, K. H. (1999). "Building Design Services in a Distributed Architecture," *Journal of Computing in Civil Engineering*, 13(1), 12-22.

ISO (2003). *Industrial Automation System and Integration – Process Specification Language*, report No. 18629-11, International Organization for Standardization.

Liu, D., Law, K. H., and Wiederhold, G. (2002). "Data-flow Distribution in FICAS Service Composition Infrastructure," *Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems*, Louisville, KY.

Liu, D., Cheng, J., Law, K. H., Wiederhold, G., and Sriram, R. D. (2003). "An Engineering Information Service Infrastructure for Ubiquitous Computing," *Journal of Computing in Civil Engineering* (accepted for publication).

McKenna, F. (1997). *Object Oriented Finite Element Analysis: Frameworks for Analysis Algorithms and Parallel Computing*, Ph.D. Thesis, Department of Civil and Environmental Engineering, University of California, Berkeley, CA.

Peng, J., and Law, K. H. (2002). "A Prototype Software Framework for Internet-Enabled Collaborative Development of a Structural Analysis Program," *Engineering with Computers*, 18(1), 38-49.

Peng, J., Liu, D., and Law, K. H. (2003). "An Engineering Data Access System for a Finite Element Program," *Advances in Engineering Software*, 34(3), 163-181.

Pitt, E. and McNiff, K. (2001). *java™.rmi: The Remote Method Invocation Guide*, 1stEd., Addison-Wesley, Boston, Ma.

Pope, A. (1998). *The CORBA Reference Guide: Understanding the Common Object-Request Broker Architecture*, 1st Ed., Addison-Wesley, Boston, MA.

Roy, J., and Ramanujan, A. (2001). "Understanding Web Services," *IT Professional*, 3(6), 69-73.

Wiederhold, G. (1996). "Strategic Uses of Information Technologies," *Seminar at Stanford Graduate School of Business*, Stanford, CA.