A Continuous Auditing Web Services (CAWS) Model for XML Based Accounting Systems

Uday S. Murthy  
Professor & Quinn Eminent Scholar of Accounting Information Systems  
School of Accountancy  
University of South Florida  
Tampa, FL 33620-5500  
Phone: 813-974-6523  
Email: umurthy@coba.usf.edu

S. Michael Groomer  
Professor of Accounting and Information Systems  
Department of Accounting & Information Systems  
Kelly School of Business  
Indiana University  
Bloomington, IN 47405-1701  
Phone: 812-855-4026  
Email: groomer@indiana.edu

University of Waterloo Symposium on Information Systems Assurance  
October 31-November 1, 2003

Authors contributed equally to the paper. Murthy won the coin toss. A previous version of this paper was presented at the Fifth Continuous Assurance Symposium at Rutgers University, November 2002.
A Continuous Auditing Web Services (CAWS) Model for XML Based Accounting Systems

ABSTRACT

This article discusses how emerging information technology (IT) frameworks such as XML and Web Services can be utilized to facilitate continuous auditing for the next generation of accounting systems. Relying on a number of components of Web services technology, this article presents a new model for continuously audit business processes, referred to as Continuous Auditing Web Service (CAWS). The CAWS mechanism would run as a “web service” in the audit firm’s computing environment and could be applied at a very granular level to provide assurance about specific business processes, at a very aggregate level for providing assurance relating to continuously reported earnings, or to provide continuous assurance about the operation of internal controls resident in the audit client’s environment. The primary user of CAWS, given the current audit model, is the audit firm itself. However, the proposed CAWS approach facilitates a new “pull” model of auditing as envisaged by Elliott (1992), where assurance consumers invoke the CAWS routines to obtain assurance on demand. In such a model, the auditor would offer restricted views provided by the CAWS routines on a fee basis to analysts, investors, financial institutions, and other parties interested in obtaining continuous assurance of business performance or other audit objects of interest. The frameworks and technologies that facilitate such a Web service based continuous auditing mechanism in an XML-enhanced world are briefly described. The article concludes with suggestions for future research.

Key words: Continuous auditing, XML, schema, web services.
I. INTRODUCTION

Advances in information technology (IT) over the last several years have fueled the move towards Web-based models of computing. Internet applications, accessible using ubiquitous easy-to-use Web browsers, proliferate for online banking, travel reservation systems, and in a multitude of other consumer and business-to-business contexts. Even enterprise resource planning (ERP) systems that handle the bulk of the internal information processing needs of large corporations are now evolving to the web model, doing away with the need for proprietary graphical user interfaces on client machines. Unlike traditional tightly coupled models such as Common Object Request Broker Architecture (CORBA) and Distributed Component Object Model (DCOM), in the web model the client (browser) and the server (web server) are loosely coupled\(^1\). This characteristic of the web model explains why it has become the most popular model of distributed computing.

The web model of computing is now on the threshold of a new era—that of “web services” that leverage the power of the eXtensible Markup Language (XML) and related technologies. Web services are loosely coupled reusable software components that encapsulate business logic and contain standardized interface mechanisms to allow their external interactions to be programmatically accessed over standard Internet protocols. Wolter (2001) defines XML web services in terms of its three essential elements, as follows:

- XML Web Services expose useful functionality to Web users through a standard Web protocol. In most cases, the protocol used is Simple Object Access Protocol (SOAP).
- XML Web services provide a way to describe their interfaces in enough detail to allow a user to build a client application to talk to them. This description is usually provided in an XML document called a Web Services Description Language (WSDL) document.
- XML Web services are registered so that potential users can find them easily. This is done with Universal Description, Discovery, and Integration (UDDI).

---

\(^1\) Distributed systems using CORBA or DCOM require that all pieces of an application be deployed at once and necessitate centralized coordination. In the loosely coupled scenario of Web-based technologies, clients and servers can easily be added as needed requiring only the centralized registration of DNS names, with a high resulting degree of interoperability, scalability and manageability.
Although the aforementioned technologies of SOAP, WSDL, and UDDI are still in flux, it appears quite likely that the next wave of business computing systems will embrace the notion of XML Web services to some degree. Consequently, future accounting systems are likely to be built using XML technologies, incorporating both a closed (proprietary) set of XML tags for internal reporting purposes and open XBRL tags for external reporting. The proposed XBRL GL taxonomy offers a mechanism for XML-based corporate accounting systems to interface with one another, regardless of differences in the XML tag set employed within each system. While this emerging landscape of XML-driven accounting systems poses significant challenges for accountants and auditors, it also provides opportunities to reengineer how a continuous auditing mechanism operates. In particular, using XML Web services technology, the continuous auditing mechanism can actually reside within the auditor’s systems rather than the auditee’s systems.

The purpose of this paper is to present a Continuous Auditing Web Services (CAWS) model that uses the emerging XML Web services framework to support a “pull model” of continuous auditing in a world of XML-enabled accounting systems. The objectives of this paper are to (1) describe and briefly discuss the major components of the emerging XML Web services framework, including the continuous auditing implications of each component, (2) discuss the criteria and requirements of a continuous auditing mechanism in an Internet-dominated computing environment, (3) describe in some detail the proposed CAWS approach to continuous auditing and how it would operate in the context of a hypothetical sales verification system, and (4) to present some issues and questions for future research in this arena. Previous research has not addressed the relationship between web services, XML/XBRL, and continuous auditing. This paper attempts to fill that void and presents an approach that facilitates a new “pull model” of auditing, where assurance customers could invoke the CAWS routine resident on the auditor’s system on demand, with micro-payments to the auditor with every invocation.

The remainder of the paper is organized as follows. Section II provides some background and describes the various components of the emerging XML Web services framework. Thereafter, section III discusses the criteria and requirements for a continuous auditing mechanism using standard Internet
protocols. Section IV presents a the CAWS approach to implementing a continuous auditing mechanism in an XML context, describing the specific components of the XML Web services framework that would be involved in such a mechanism. An illustration in the context of a sales verification system is also presented in section IV. The concluding section summarizes the paper and discusses future directions in this line of research.

II. BACKGROUND

The term “Web services” is extremely generic and lacks a widely accepted definition. The official World Wide Web Consortium (W3C) definition follows (W3C, 2002):

“A Web Service is a software application identified by a URI [IETF RFC 2396], whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols.”

The term “Web services” can be thought of as a set of technologies, specifically XML, WSDL, SOAP, and UDDI that collectively define the concept. Conceptually, Web services can be thought of as a “stack” of emerging standards that describe a service-oriented, component-based application architecture. In this emerging model, each “web service” is a stand-alone discrete e-business process that is accessible from anywhere on the network (i.e., the Internet). Each web service has internal enterprise-specific private business logic that is not exposed to the Internet and also public interfaces that allow other web services of business partners to exchange data on demand without requiring any customization on the part of any of the web services that so interact.

The component-based notion of Web services draws from the object-oriented paradigm, where blocks of code can be reused and extended in new ways. Similar to the applet concept in Java, each web service performs one or more tasks and describes its interfaces so that other applications, which could be web services themselves, know how to invoke the service’s functionality. The “loose coupling” aspect of Web services implies that developers do not need to “hard code” the intricate details required for end-to-end communication between web services. Rather, the interfaces work much like the interactions between web browsers and web servers—they are seamless, platform neutral, and transparent to both the web
server and the browser. Thus, the goal of a standards-based XML Web services framework is to provide a platform for building distributed applications using software running on different operating systems and devices, written using different programming languages and tools from multiple vendors, developed and deployed independently, and yet with seamless interoperability. A listing of publicly available web services is available at http://www.xmethods.com.

How web services work

As indicated, web services build on the loose coupling of the traditional Web programming model, and extend it for use in other kinds of applications. XML Web services differ from traditional Web applications in three ways: (1) Web services use SOAP messages instead of Multipurpose Internet Mail Extensions (MIME) messages, (2) Web services are not HTTP-specific, and (3) Web services provide metadata describing the messages they produce and consume.

SOAP

SOAP is a protocol for messaging and communication between applications, including error messages. Web services communicate using SOAP messages. SOAP is based on XML and uses common Internet transport protocols (i.e., HTTP) to carry its data. Developed initially by Microsoft and other vendors, SOAP is now in the hands of the World Wide Web Consortium (W3C) standards body and will eventually be released as “XML Protocol (XMLP).” SOAP toolkits are currently available from a number of vendors, but there is considerable variance in the types of function calls and the data types of the parameters supported. Since it has its roots in XML, SOAP is extensible. Enhancements to the SOAP protocol to provide message integrity, message confidentiality, and message authentication are already in the works under the label Web Services Security, or WS-Security. The body of a SOAP message contains whatever XML an application wants to send. A sample SOAP message is shown in Appendix 1.

A key difference between an XML-based SOAP message and a traditional MIME-typed message is that the client application (i.e., the web browser) merely displays the HTML page, whereas the web

---

2 Microsoft SOAP toolkit, Apache SOAP, and NuSOAP are three examples of SOAP toolkits. Web sites for these and other tools are provided in Appendix 1.
service client must interpret the data in the XML message and perform some action based on the data. Whereas standardized HTML tags specify the formatting of data on a web page, XML tags are customized to provide a standard way of representing the contents of data on a web page and are therefore a logical choice as a message format for Web services. The SOAP protocol mandates the use of XML to represent the message, but the actual message content depends on the purpose and corresponding design of the Web service.

**WSDL**

For the exchange of web services XML messages to function seamlessly, there must be a structured standardized language for describing the communication. WSDL addresses this need by defining an XML grammar for describing Web Services as collections of communication endpoints that are capable of exchanging messages. WSDL service definitions thus serve as a recipe for automating the details involved in application communication between Web services. The notation used for the service and message format definitions in a WSDL file is based on the XML Schema standard. In addition to describing message contents, WSDL is also used to define the network location of the service and the communications protocol for interfacing with it. Thus, the WSDL file contains all the necessary requirements for writing an application to work with an XML Web service. Many SOAP toolkits also provide support for generating WSDL files from existing program interfaces. However, few tools exist for directly creating WSDL files. Microsoft’s Visual Studio®.NET provides the functionality necessary for reading a WSDL file and generating the code required to communicate with an XML Web service. Further technical details regarding WSDL are contained in Appendix 1.

**UDDI**

The Universal Description, Discovery, and Integration (UDDI) specification defines a SOAP-based Web service for locating Web services and programmable resources on a network. UDDI is simply the yellow pages of Web services. Each entry in the UDDI directory is an XML file that provides information about a company and the Web services it makes available. The three parts to an entry in the UDDI directory are (1) “white pages” that describe the company offering the service: name, address,
contacts, etc., (2) “yellow pages” that include industrial categories based on standard taxonomies such as the North American Industry Classification System and Standard Industrial Codes, and (3) “green pages” that provide details of the interface to the service, for use by designers writing an application that uses the Web service. In UDDI, services are defined in a document called a “Type Model” or tModel, which typically contains a WSDL file that describes a SOAP interface to an XML Web service. Appendix 2 lists Internet resources relating to the aforementioned technologies relating to XML web services.

**Business Process Recording using XML**

While SOAP, WSDL, and UDDI represent the core set of technologies to make Web services work, none of these components specify how the content of business process messages should be structured. There is no standard “XML grammar” for defining and structuring business process messages. At present, there are at least two alternatives aimed at filling this void: XBRL GL and BPEL4WS.

**XBRL-GL**

From its origins in 1998, the eXtensible Business Reporting Language (XBRL) initiative is now backed by a consortium of about 170 companies. While XBRL comprises an XML based standard for external financial reporting, the related XBRL General Ledger (XBRL GL) specification is aimed at internal accounting systems at the transaction level. Referred to as the “Journal taxonomy,” the XBRL-GL specification can be used for representing both financial and non-financial information. It is both extensible, given its roots in XML, and standards-based enabling cross-platform information exchange around the globe. The XBRL GL “core” contains approximately 50 fields, providing sufficient functionality for representing any accounting transaction as well as standard ledger account information for maintaining accounts receivables, payables, inventory, etc. A related development is the XBRL IFRS (International Financial Reporting Standards) initiative for a common international XML tag set for financial reporting (Ramin and Prather 2003). The XBRL IFRS specification is intended to integrate with XBRL GL.

The official XBRL GL site executive summary lists the following features unique to the specification:
• “XBRL GL is chart of accounts independent. It does not require a standardized chart of accounts to gather information, but it can be used to tie legacy charts of accounts and accounting detail to a standardized chart of accounts to increase communications within a business about what needs to be measured and why.

• XBRL GL is reporting independent. It collects general ledger and after-the-fact receivables, payables, inventory and other non-financial facts, and then permits the representation of that information using traditional summaries and through flexible links to XBRL for reporting. As XBRL GL does not assume financial reporting or any specific type of output, it becomes an important repository for future metrics such as ValueReporting. Systems to do ValueReporting can reduce their development time using XBRL GL as part of their development process.

• XBRL GL is system independent. Any developer can create import and export routines to convert its information to XBRL GL format, or our firm can help develop tools to do so. This means that accounting software developers need only consider one design for their XML import/export file formats. Application service providers (ASPs) can offer to supply XBRL import and output so end users can more easily use their own data. Companies developing operational products, such as point of sale systems or job costing, or reporting tools can link with many accounting products without needing specialized links to each one.

• XBRL GL is based on XML. XML is the future of data, as seen by recent announcements from all of the major software developers. The openness and power of XML will enable new products and services, and make possible new management real time dashboards, as well as the future of tools such as continuous audit 'bots, which monitor the accounting data streams from various places, with triggers and alarms for auditing by exception.

• XBRL GL permits consolidation. Popular low-end products, like Quickbooks, and mid-market solutions are not designed to facilitate consolidating data from multiple organizations. XBRL GL can help transfer the general ledger from one system to another, be used to combine the operations of multiple organizations, or bring data into tools that will do the consolidation.

One use of XBRL GL for a company is the creation of a “data hub” that receives XML-based inputs from the company’s internal accounting system and provides XML-based outputs to systems within or outside the company. For example, such a data hub could be used by a company with subsidiaries in different countries and/or subsidiaries with incompatible internal accounting systems to simplify the inter-company exchange of data. Externally, the data hub could be used by a company to simplify the process of XML information interchange with its business partners. As discussed later in the paper, a key use of such a data hub could be to service SOAP requests coming from a continuous auditing web service that resides in the auditor’s environment.

4 It should be noted however, that there are a number of competing specifications for XML-based data transfer between business partners, specifically ebXML and the X12 XML Reference Model.
A consortium⁵ of companies has recently proposed a notational model for specifying business process behavior based on Web Services. This model is called Business Process Execution Language for Web Services—BPEL4WS (IBM 2003). The BPEL4WS process model is layered on top of the service model defined by WSDL 1.1. Per the latest specification of BPEL4WS, its purpose is defined as follows (IBM 2003):

“BPEL4WS provides a language for the formal specification of business processes and business interaction protocols. By doing so, it extends the Web Services interaction model and enables it to support business transactions. BPEL4WS defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.“

The purpose of the BPEL4WS process model is to specify peer-to-peer interaction between services described in WSDL. Both the process of interaction and the partners involved are modeled as WSDL services. In essence, a BPEL4WS process definition provides and/or uses one or more WSDL services, and provides the description of the behavior and interactions of a process instance relative to its partners and resources through Web Service interfaces. That is, BPEL4WS defines the message exchange protocols for a specific role in a business process interaction.

In BPEL4WS terminology, business processes can be either executable or abstract. Executable business processes model actual behavior of a participant in a business interaction, without separating or distinguishing between externally visible (“public”) aspects of a business process and specifics of its internal workings. Abstract business processes are essentially business protocols that use process descriptions specifying the mutually visible message exchange behavior of each of the parties involved in the protocol, without revealing their internal behavior. Abstract processes are not executable and are meant to couple Web Service interface definitions with behavioral specifications, which are used to precisely define the behavior of each party involved in a business protocol. Both executable and abstract business processes can be defined using BPEL4WS. However, the feature sets for data handling differ

---

⁵ As of late September 2003 the consortium consisted of IBM, Microsoft, SAP, BEA, and Siebel Systems.
depending on whether BPEL4WS is being used for defining executable or abstract business processes. The BPEL4WS syntax for defining executable business processes is categorized into the following four major sections: containers, partners, fault handlers, and a main processing section.

A compelling aspect of BPEL4WS is that it is extensible, given its roots in XML. Constructs used in a BPEL4WS document can be extended by referencing constructs from other XML namespaces. So long as the constructs from extension namespaces do not conflict with BPEL4WS constructs, the language can be made robust enough to handle the business process definition needs for even the most unique of situations. The structure and syntax of the BPEL4WS language is shown in Appendix 1.

At this point, there are two paths that could be followed to realize a web service based approach to continuous auditing. One is to custom develop an internal accounting system using BPEL4WS as the underlying XML-based language. Such a BPEL4WS based system would be ideally suited to handle the processing requirements of the continuous auditing web services model outlined later in the paper. The second path involves the creation of XBRL GL data hubs, which would serve as the intermediary between the internal accounting system of the target company and the auditor’s system where the continuous auditing web service resides. However, the latter approach involves an additional layer of translation between the target company’s internal accounting system (whether XML based or not) and the XBRL GL data hub. Automated translators do exist to facilitate the process. The next section discusses the criteria and requirements for continuous auditing.

**III. CRITERIA AND REQUIREMENTS FOR CONTINUOUS AUDITING**

Individuals and entities interested in a company’s financial statements and other information objects seem to have an insatiable appetite for information delivered to them sooner rather than later. The growth of the World Wide Web and the tools that support communication in this environment have provided an infrastructure to satisfy these appetites. Moreover, many organizations have overhauled their information systems during the last two decades primarily in response to the Year 2000 problem and the inability of aging legacy systems to provide high quality information for decision making. This work has involved the installation of competent general ledger systems and in many instances substantial
reengineering efforts supported by ERP systems. These developments have set the stage for continuous online reporting and bringing with it, the need for continuous auditing on these timely delivered financial statements and other information objects. Continuous auditing has been articulated by the CICA-AICPA Joint Study Group (Continuous Auditing, 1999, p. 5).

“A continuous audit is a methodology that enables independent auditors to provide written assurance on a subject matter using a series of auditors’ reports issued simultaneously with, or a short period after, the occurrence of events underlying the subject matter.”

Note that this definition embraces the notion that “shortly after” the client information becomes available, the auditor must be positioned to provide written assurance on the information in question. This definition also assumes that the client’s information systems are positioned and calibrated to provide the necessary information that will be the subject of audit. This definition uses the wording “independent auditor.” It is important to note that the independent auditor may be external or internal to the client. Although the definition specifies that the assurance provided be written, it does not specify the precise form of such “written” assurance. Further, the definition assumes a “push” model of delivering continuous assurance, where the auditor delivers written assurance simultaneously or shortly after the occurrence of events of interest.

In the XML environments, it is clear that auditors could use XML based systems to conduct a variety of audit work, including continuous auditing as envisioned by the CICA-AICPA Joint Study Group. But the continuous audit process described by the joint study group involves a number of challenges that the CAWS process proposed in this paper overcomes. The XML Web services technology discussed in the current paper uniquely envisions a “pull” model of continuous assurance, where consumers desirous of obtaining assurance can (continuously) retrieve assurance reports over the Internet.

**Conditions for a continuous audit**

In order to conduct a continuous audit a number of conditions must be present. These conditions are that
• the client must have highly reliable systems. These systems must be able to provide the necessary subject matter to the auditor on a timely basis.

• the subject of the audit has suitable characteristics necessary to conduct the audit. For example, if the audit is focused on evaluating internal controls, then the auditor must be able to electronically interrogate these controls.

• the auditor must have a high degree of proficiency in information systems, computer technology, and the audited subject matter.

• automated audit procedures will provide most of the audit evidence necessary to opine on the subject of the audit.

• the auditor must have a reliable means of obtaining the necessary audit evidence so that an opinion can reached.

• the auditor must have timely access to and control over any audit evidence generated as a result of the continuous auditing procedures.

• it is necessary to have a “highly placed executive” in the client organization who will serve as a champion for the adoption and support of continuous auditing.

Certain additional conditions pertaining specifically to XML must also be met. The audit client’s accounting system must be XML compliant, XML Web Services technology must be implemented within the audit firm’s environment, and the technologies discussed in this paper (BPEL4WS, SOAP, WSDL) must mature and stabilize to the point of high reliability.

The continuous audit process

Previously published research dealing with methods for facilitating continuous auditing has focused on the use of or variants of Embedded Audit Modules (EAMs) (see Groomer and Murthy, 1989; Vasarhelyi and Halper, 1991; Continuous Auditing, 1999; Groomer and Murthy, 2003). Groomer and Murthy (1989) discuss how EAMs can be employed in a database environment. Vasarahelyi and Halper (1991) present a continuous auditing implementation in a corporate setting. Regardless of the exact mechanisms used to facilitate continuous auditing, the objective of the audit process remains the same. That is, the auditor must gather enough competent evidential matter such that the achieved audit risk is at a level acceptable to the auditor.

As an illustration of continuous auditing in the Internet age and before the advent of XML Web services, consider a scenario where a client publishes financial statements to the corporate website on a
monthly basis. Responding to some force, the client has requested that the external auditor opine on these monthly web site financial statements. The audit process necessary to conduct such work would entail the following. First, the fiscal year end would begin with the assumption that the previous years’ financial statement received an unqualified opinion. Also, the internal controls for this client are assumed to be of high quality and that control risk is assessed low. In order to opine on the financial statements being published to the website on a monthly basis, the auditor would employ primarily test of controls and analytical procedures to substantiate the fair presentation of the financial statements and would employ continuous auditing techniques like EAMs to gather the necessary evidence. Moreover, on a quarterly basis, one might assume that the auditor would, in addition to using EAMS, employ more traditional end of period substantive procedures to ascertain whether or not the financial statements were fairly stated. In this example, the traditional test of transaction audit objectives like existence, completeness, accuracy, classification, timing, etc. are used to drive these monthly assessments of the financial statements. Put another way, nothing has really changed the underlying focus for the audit work. Only the techniques and methods for gathering the evidence have changed. However, as we discuss in the next section, the change stemming from a move to Web Services technology to facilitate continuous auditing is quite dramatic.

Limitations of conventional continuous auditing techniques

The use of continuous auditing techniques (CATs) like EAMs suggests a number of detractions. These issues are discussed below.

Continuous audit techniques (CATs) like EAM’s or ITF are constructed by embedding them in a client’s application system. Thus, it is important that client side general controls be put into place to protect and prevent any tampering with the audit code by client personnel. Conventional CATs typically require extensive modifications to the client’s systems, which (1) make them extremely expensive to implement post-hoc and (2) are likely to be resisted by the client for a variety of reasons. For example, the client may be concerned that the CATs might cause their system to become unstable; errors in auditor-installed CATs might bring down the client’s system; or the auditor has access to proprietary code developed by the client. The major concern here is that the EAM or ITF techniques are typically viewed
as involving modifications to the client’s system. Unless and until the auditor can demonstrate that the traditional CATs will not have a negative impact on the client system, the client will be very reluctant to provide the auditor with the access needed to install a CAT.

Conventional CATs are “brittle” in that changes made to the client’s system may require changes to the CATs, further exacerbating the problems of high cost of the CATs and the likelihood that the errors in the CATs might negatively impact the client’s production systems. Conventional CATs consume resources on the client’s systems. There may be response-time issues in high volume transaction processing systems, if CATs are operational continuously (Groomer and Murthy 2003). Conventional CATs would appear to be an expensive undertaking in many settings and applications. Expensive here would be defined both in terms of time and money.

The CAWS model of continuous auditing, described in the following section, overcomes the aforementioned limitations of conventional CATs.

IV. CONTINUOUS AUDITING USING WEB SERVICES FRAMEWORK COMPONENTS

With XML Web Services as a facilitator for continuous auditing, conventional techniques such as EAMs and software agents lodged within the client’s computer system are no longer applicable. Rather, continuous auditing functionality is defined as a set of Web Services that reside within the auditor’s computing environment rather than within the auditee’s computer system. We coin the term Continuous Auditing Web Service (CAWS) to describe how continuous auditing would operate in an XML-enhanced business processing environment. In this scenario, users requesting assurance on client business processes on a continuous or frequent basis do so by invoking auditee-specific CAWS residing within the auditor’s computing environment.

The use of CAWS can be envisioned in much the same manner as CATs (EAMs and ITF). That is the external auditors would use CAWS to assist in the gathering of audit evidence to support an opinion on an audit client’s financial statements. Another service and resulting revenue stream that could be available to the external auditor is by implementing a new “pull” model of auditing, as suggested by Elliott (1992). In such a model, the external auditor could provide assurance services on any number of
client objects (revenue streams, internal controls, etc.). Users (stockbrokers, analysts, investors, bankers, etc.) seeking this information and related assurance could access the objects and related assurance from the auditor provided CAWS. Each time the CAWS is invoked, the auditor receives a fee for the service. While the auditor has full scope to access client data in an opinion audit, the auditor would be providing specific information products (more limited views) related to the client with client permission. This process brings an entirely new business model and different dimension for the work of the public accounting firm. In effect, the CAWS model described in this paper represents one possible approach to implementing Elliott’s (1992) vision of end users acquiring on-demand access to report objects and their accompanying opinion for a fee.

Figure 1 depicts the CAWS model, indicating how the continuous audit web service resident on the auditor’s system interfaces with WSDL wrappers for each business process within the audit client’s accounting system.

In terms of BPEL4WS terminology, the WSDL wrapper for each business process includes CAWS-specific portType mappings to facilitate SOAP communication using the HTTP protocol between the auditor and the auditee. Not only can external auditors use the CAWS to facilitate the external audit process, but the CAWS can be invoked on demand by continuous assurance (CA) customers such as investors, analysts, and financial institutions, i.e., end users requiring attestation of business process assertions being made by the auditee or other audit objects of interest. For the external auditor, the use of CAWS represents a potential revenue stream not previously considered. The continuous audit relevant reporting provided by CAWS is rendered via conventional Web pages, by applying an XSLT style sheet to the XML CAWS audit report data. Given that the CAWS reside in the auditor’s environment and not the auditee’s environment, the proposed framework serves to reinforce auditor independence.

As discussed in the previous section, an alternative to using BPEL4WS for defining business processing routines is to convert the outputs of existing accounting systems to XBRL GL. Such an
approach would involve the creation of an XBRL GL “data hub” to which all internal accounting subsystems feed data. The data hub would interface with WSDL wrappers defined for the CAWS, to facilitate the continuous auditing approach proposed in this paper. Figure 2 depicts such an alternative model.

BPEL4WS and CAWS

While BPEL4WS and XBRL GL represent alternative approaches that are not necessarily incompatible, we focus on the BPEL4WS approach for the remainder of the paper. We now consider some of the mechanics of how the architecture depicted in Figure 1 could be driven to the point of a functioning continuous assurance system in an XML world. As shown in Figure 1, the various business processes within each set of related activities commonly thought of as transaction cycles (i.e., revenue, procurement, etc.) are defined in BPEL4WS syntax, enabling communication between applications within the auditee’s environment as well as communication with external parties via portTypes. Communication within applications facilitates a key goal of the implementation of ERP systems, i.e., cross-functional integration. The auditee-specific activities within each transaction cycle are defined using executable BPEL4WS business processes. Executable business process, per BPEL4WS syntax, are private—they remain hidden from external Web Services. For each cycle, the WSDL wrappers are essentially abstract BPEL4WS business processes, which, as indicated earlier, specify only the public aspect of the business process, i.e., how the process interacts with Web services outside the auditee environment. It is within the WSDL wrapper for each business process that the business process specific parameters of interest to the auditor’s CAWS are made available for access via portTypes.

Each business process specific WSDL wrapper, written in BPEL4WS, specifies the auditor information in the “partners” section, with references to the auditor CAWS that reference that particular business process. Within the auditor’s environment, a CAWS must be created for each business process, defining the audit client’s business process parameters that must be “called” via portTypes.
“bindings” section within each CAWS must exactly match what is specified in the client business process WSDL wrapper. The <input> operation of each CAWS portType must be fed by the <output> portType of the client business process that is being “checked” by the auditor’s CAWS. This manner of operation is analogous to how generalized audit software (GAS) such as ACL operates. For instance, using GAS at the year-end audit, the client’s data files “feed” auditor developed programs in ACL. These auditor-developed programs then perform audit procedures in a computing environment controlled by the auditor (i.e., the auditor’s own computer) to generate outputs aimed at ascertaining whether various audit objectives have been achieved. Many GAS packages can interface directly with the client’s accounting system via ODBC to seamlessly retrieve transaction data, with the audit software routines running on the auditor’s computer.

Each CAWS must then define “faulthandlers” for audit exceptions, in terms of activities performed relative to the business process specific parameters retrieved via SOAP/HTTP. The typical action in reaction to a fault (an exception of audit interest) would be to log the exception, although other actions such as email notification might be warranted for serious exceptions. Data about the exceptions would be stored in data structures defined within the “containers” section of the CAWS. When the CAWS resident in the auditor’s environment are invoked by the CA consumer, the CAWS would (1) access the business process specific parameters, (2) process CAWS activities, with fault handlers being triggered for exceptions, (3) store exception data in “containers” for each CAWS, and (4) retrieve container data and report the results to the end user, i.e., the assurance requested by that user. What constitutes an “exception,” the business process specific parameters are needed to determine whether exceptions exist, and the data to be stored in the CAWS containers would all vary as a function of the type of business process and the audit objectives relative to the business process, in terms of existence, completeness, and accuracy of transaction processing.

**A Sample CAWS Specification**

The CAWS model of continuous auditing in an XML-enhanced computing world is now described in the context of a hypothetical scenario. Consider an XML-enabled e-commerce sales
transaction processing application for an auditee. One aspect of verification demanded by assurance customers is whether the revenues being (continuously) reported by the audit client are “valid,” i.e., whether there is underlying support for the sales transactions that constitute the revenue number being reported. Figure 3 depicts how a sales verification CAWS would work to provide continuous assurance regarding the revenue amounts being reported by the audit client.

Following the model laid out in Figure 1, the process begins with a request for assurance by some CA consumer (investor, analyst, financial institution, etc.). The request is made to the auditor’s system by some CA customer, invoking the sales verification CAWS in the auditor’s system. The sales verification CAWS in turn retrieves data from the audit client’s sales system based on parameters defined at the time the CAWS was invoked by the CA consumer. The data retrieved by the CAWS from the sales system comprises fields defined as output port types in the WSDL wrapper of the sales system, as shown in Figure 3. Upon receipt of the data items from the sales system, the sales verification CAWS in the auditor’s system then requests confirmatory data from reference systems, as follows: order related data from the client’s sales order system (were sales orders received?), credit related information from the client’s credit approval system (do customers have approved credit?), inventory related data from the client’s inventory system (are the products valid and are they priced correctly?), shipping related data from the client’s shipping system (was the merchandise shipped?), and invoicing related data from the client’s billing system (was the sale billed to the customer?). In essence, these requested data items serve the purpose of verifying the particulars of the sales transactions being reported by the client’s sales system. Based on the responses received from the client’s reference systems, the sales verification CAWS in the auditor’s system does the necessary processing (matching, calculations, etc.) and returns an assurance result (positive or negative) to the CA customer.

---

6 For example, one parameter might be the time frame for which assurance is requested. Another parameter might be the level of detail of the CAWS assurance report.
A potential extension to the model depicted in Figure 3 is for the auditor’s CAWS to request confirmatory data from the appropriate external agents responsible for the execution of specific sub-processes within the overall sales business process for the audit client. For the sales verification scenario depicted in Figure 3, confirmatory data could be sought by directly polling the systems of entities in the audit client’s supply chain. Figure 4 depicts how the process might operate.

As indicated in Figure 4, the extension can be thought of as “real time confirmations” performed by the auditor, specifically, the CAWS residing in the auditor’s system. To verify the validity and completeness of the sales figures being reported by the audit client, confirmatory data could be automatically sought by the CAWS, as follows: polling the audit client’s customers’ purchasing systems (“Did you purchase the merchandise the client claims to have sold you?”), polling the audit client’s shippers’ systems (“Did you ship the merchandise the client claims to have shipped?”), and polling the audit client’s suppliers’ sales systems (“Did you supply to the audit client merchandise that the client claims to have sold?”).

The CAWS model in Figure 4 that extends across the supply chain would of course need an extensive level of standardization and cooperation throughout the supply chain. However, all parties in the supply chain would benefit from such a supply-chain-wide CAWS model. Especially for large companies such as Wal-Mart that interact with numerous large suppliers, such a model would be economically justifiable for both Wal-Mart and its suppliers. For example, consider the case where Proctor and Gamble sells merchandise to Wal-Mart. Verification of sales made by Proctor and Gamble to Wal-Mart equate to verification of purchases made by Wal-Mart from Proctor and Gamble. Assurance customers of both Wal-Mart and Proctor and Gamble would therefore benefit from a CAWS sales/purchases verification system implemented by the auditors of both companies. Further, a single CAWS implementation for Company X could service continuous assurance requests from customers seeking assurance for any number of companies up and down the supply chain for Company X.
Level of application of CAWS model

The above discussion has focused on how the CAWS “pull” model operates in the context of providing continuous assurance at very granular (disaggregated) level, i.e., continuous assurance regarding a specific business process at the target entity (e.g., audit client’s sales transactions). At the most aggregated level, the CAWS model would be employed to provide continuous assurance to investors regarding earnings reported by the auditee, presumably in an environment where earnings are reported on a continuous basis. For the CAWS model to operate at such an aggregate level, in addition to routine transaction data, the auditor’s CAWS would have to interface with the client’s system to retrieve data relating to periodic adjustments and allowances, prorated for the number of days in the reporting period. The CAWS processing relating to such prorated adjustments would likely include analytical procedures to verify the reasonableness and appropriateness of the adjustments. However, given the “soft” nature of these prorated adjustments and allowances, the CAWS assurance regarding continuously reported earnings will necessarily include caveats regarding the imprecision of the earnings figure.

Another target for assurance to which the CAWS model could be applied is to verify that the internal controls implemented within the audit client’s systems are functioning properly. Operating somewhat like a continuous SysTrust verification system, the CAWS would be configured with a series of test transactions designed to verify the proper operation of internal controls within the audit client’s system. The suite of test transactions configured within the control verification CAWS would be similar to what would be programmed into an integrated test facility, also known as Systems Control Audit Review File (SCARF), aimed at verifying the operation of controls at random points in time throughout the audit period (Weber 1999). Upon being invoked by an assurance customer, the control verification CAWS would send test transactions to the client’s system, which would respond with either positive or negative confirmation indicative of whether controls are operational or not operational. In this manner,

---

7 For example, for a company with a 12/31 year ending, if the earnings assurance CAWS module is invoked on May 25, all adjustments and allowances will have to be applied for 145 days on a prorated basis.
the control verification CAWS could be invoked by users whenever assurance about the functioning of internal controls in the audit client’s environment is sought.

The limitations of conventional CATs (EAMs and ITF) described previously are to a large extent not experienced when CAWS is considered. Relative to EAMs and ITFs, the use of CAWS is a significantly less invasive procedure in terms of the changes that need to be made to the client’s systems. CAWS operates using the constructs previously described in a web-based open systems environment. In contrast to inserting extensive sections of code into the client’s systems, as is necessary with EAMs and ITFs, the CAWS approach requires only that the client’s XML-based systems define the necessary “port types” within WSDL wrappers to facilitate SOAP communication with the CAWS resident on the auditor’s system. A key advantage of CAWS relative to conventional CATs, is that the bulk of processing is done on the auditor’s system rather than on the client’s system. Thus, the concern regarding resource usage on the client’s systems is mitigated (Groomer and Murthy 2003). Of course, as with any systems work, there is a relatively high initial cost involved in making CAWS work. However, it would not, in our opinion, be as costly as the more traditional CAT counterparts. As with traditional CATs, the benefits of the CAWS implementation would be reaped in years after the initial year of implementation, assuming the client’s systems are sufficiently stable obviating the need for constant major changes to the CAWS.

V. SUMMARY AND CONCLUSION

This article discussed how emerging IT frameworks such as XML and Web Services can be utilized to facilitate continuous auditing for the next generation of accounting systems. The components that comprise XML Web Services technology (SOAP, WSDL, and UDDI) were described. The recently proposed Business Process Execution Language for Web Services was then discussed. After indicating the requirements and criteria for a continuous auditing system in an XML enhanced world, the article proposed an architecture for accomplishing a “pull model” of continuous auditing where end users “call” Continuous Auditing Web Services that reside in the auditor’s computing environment. Thus, the continuous auditing mechanism itself runs as a “Web Service” with all the advantages stemming from this emerging technological framework.
A logical next step in this line of research is to implement a prototype system demonstrating the feasibility of the CAWS architecture as articulated in this paper. The prototype could be developed following either the BPEL4WS route or the XBRL GL route. In addition to demonstrating the feasibility (or lack thereof) of the CAWS approach, the development of a realistic prototype might also reveal whether any extensions are necessary in the BPEL4WS and/or WSDL specifications to accommodate the unique needs of continuous auditing. As noted earlier, being based on XML, both BPEL4WS and WSDL are inherently extensible. While the development of a prototype system is clearly a first step, future research could explore a host of related issues such as the computational demands imposed by CAWS, how “continuously” the CAWS can be invoked, how a “pull model” of continuous assurance is received by end users, and how much end users are willing to pay every time the CAWS service is invoked to obtain assurance. Future research could also more thoroughly investigate how the CAWS model would operate to provide continuous assurance regarding continuously reported earnings.

It seems relatively certain that accounting systems are moving into an arena where the lingua franca of business data processing is XML. In an increasingly wired world and a global economy, information delivered anywhere at anytime is no longer futuristic—it is a present day reality. Given the “crisis in confidence” fueled by the dramatic collapses of Enron and Worldcom, it seems certain that investors, regulators, and creditors will increasingly demand that information about financial performance not only be delivered on a more timely basis, but that it be provided with continuous assurance by independent auditors. The framework proposed in this paper is a step in the direction of meeting this demand.
REFERENCES


Figure 1
The Continuous Auditing Web Service Model, Assuming BPEL4WS Processing Routines

Auditor's Continuous Audit Web Service (CAWS) -> SOAP over HTTP -> Client's Application System

Audit Firm's System

XSL Style Sheet applied to XML CAWS output

Invoked on demand

CA Customers (Investors, Analysts, Financial Institutions)

Revenue System:
Processing routines in BPEL4WS
WSDL Wrapper with CAWS-specific portTypes

Procurement System:
Processing routines in BPEL4WS
WSDL Wrapper with CAWS-specific portTypes

Payroll System:
Processing routines in BPEL4WS
WSDL Wrapper with CAWS-specific portTypes

GL system:
Processing routines in BPEL4WS
WSDL Wrapper with CAWS-specific portTypes
Figure 2
The Continuous Auditing Web Service Model, Assuming XBRL GL Data Hub

Audit Firm’s System

Audit Firm’s Continuous Audit Web Service (CAWS)

XSL Style Sheet applied to XML CAWS output

Invoked on demand

CA Customers (Investors, Analysts, Financial Institutions)

Client’s Application System

WSDL Wrapper with CAWS-specific portTypes

XBRL GL Data Hub

Revenue System

Procurement System

Payroll System

GL System
Figure 3
CAWS for Sales Verification

Client’s Sales System (target system)

- Client’s Order Taking System (reference system)
  - BPEL4WS processing routines
  - WSDL Wrapper output portTypes: <invoiceno>, <invoicedate>, <customerno>, <amount>, <productno_1>, <price_1>, <qty_1>, … <productno_n>, <price_n>, <qty_n>

- Client’s Credit Approval System (reference system)
  - BPEL4WS processing routines
  - WSDL Wrapper input portTypes: <customerno>, <orderno>

- Client’s Inventory System (reference system)
  - BPEL4WS processing routines
  - WSDL Wrapper input portTypes: <productno_1>, <price_1>, <productno_n>, <price_n>

- Client’s Shipping System (reference system)
  - BPEL4WS processing routines
  - WSDL Wrapper input portTypes: <shipmentno>, <shipperno>, <customerno>, <amount>

- Client’s Billing System (reference system)
  - BPEL4WS processing routines
  - WSDL Wrapper input portTypes: <customerno>, <amount>

CAWS Request
CAWS retrieval

Audit Firm’s System

- Auditor’s Sales Verification CAWS

CA Customers (Investors, Analysts, Financial Institutions)

Invoke CAWS

Assurance result
Figure 4
CAWS for Sales Verification:
Extension Across Supply Chain

Client’s Sales System (target system)
Appendix 1

Technical Details of SOAP, WSDL, and BPEL4WS

Sample SOAP envelope for a sales order

```xml
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Body>
    <ns:Order xmlns:ns="urn:somecompany.com:orders">
      <orderno>12345678</orderno>
      <orderdate>2002-11-02</orderdate>
      <customerID>111267</customerID>
      <item>
        <itemID>876508123</itemID>
        <itemPrice>9.95</itemPrice>
        <itemQty>40</itemQty>
      </item>
      <item>
        <itemID>876440123</itemID>
        <itemPrice>19.95</itemPrice>
        <itemQty>30</itemQty>
      </item>
      <item>
        <itemID>899308179</itemID>
        <itemPrice>10.45</itemPrice>
        <itemQty>25</itemQty>
      </item>
    </ns:Order>
  </env:Body>
</env:Envelope>
```

WSDL Key Words

<table>
<thead>
<tr>
<th>Type</th>
<th>A container for data type definitions using some type system, typically using eXtensible Schema Definition (XSD) language.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>An abstract, typed definition of the data being communicated. The &lt;message&gt; element of a WSDL file specifies the input and output parameters of operations.</td>
</tr>
<tr>
<td>Port type</td>
<td>An abstract set of operations supported by one or more endpoints.</td>
</tr>
<tr>
<td>Operation</td>
<td>An abstract description of an action supported by the Web service and is specified using the &lt;operation&gt; element. Operations can have either one or all three of the following elements: &lt;input&gt;, &lt;output&gt;, and &lt;fault&gt;.</td>
</tr>
<tr>
<td>Binding</td>
<td>A concrete protocol and data format specification for a particular port type, specifying how each &lt;operation&gt; call and response is sent over the wire.</td>
</tr>
<tr>
<td>Service</td>
<td>A collection of related endpoints and contains &lt;port&gt; elements.</td>
</tr>
<tr>
<td>Port</td>
<td>A single endpoint defined as a combination of a binding, referencing one of the &lt;binding&gt; elements in the WSDL document, and a network address.</td>
</tr>
</tbody>
</table>
A Sample WSDL File

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="FooSample"
targetNamespace="http://tempuri.org/wsdl/"
xmlns:wsdl="http://tempuri.org/wsdl/"
xmns:types="http://tempuri.org/wsdl/
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmns:soap="http://schemas.xmlsoap.org/wsdl/soap/
xmns="http://schemas.xmlsoap.org/wsdl/">
<types>
<schema targetNamespace="http://tempuri.org/xsd"
xmns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/
xmns:wsdl="http://schemas.xmlsoap.org/wsdl/"elementFormDefault="qualified">
</schema>
</types>
<message name="Simple.foo">
<part name="arg" type="xsd:int"/>
</message>
<message name="Simple.fooResponse">
<part name="result" type="xsd:int"/>
</message>
<portType name="SimplePortType">
<operation name="foo" parameterOrder="arg">
<input message="wsdlns:Simple.foo"/>
<output message="wsdlns:Simple.fooResponse"/>
</operation>
</portType>
<binding name="SimpleBinding" type="wsdlns:SimplePortType">
<stk:binding preferredEncoding="UTF-8" />
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="foo">
<soap:operation soapAction="http://tempuri.org/action/Simple.foo"/>
<input>
<soap:body use="encoded" namespace="http://tempuri.org/message" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded" namespace="http://tempuri.org/message" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
</binding>
<service name="FOOSAMPLEService">
<port name="SimplePort" binding="wsdlns:SimpleBinding">
<soap:address location="http://carlos:8080/FooSample/FooSample.asp"/>
</port>
</service>
</definitions>

BPEL4WS Sections

- Containers section: This section defines the data containers used by the process in terms of WSDL message types. Containers are essentially repositories that allow processes to maintain
state data between interacting services. The process history of messages exchanged can also be stored in containers.

- **Partners section**: This section is used to define all the parties that interact with the business process. For each business process partner a service link type and a role name must be specified. These elements identify the functionality that must be provided by the business process and by the partner to implement the Web service relationship. In particular, the functionality necessary must be defined via `<portTypes>` in WSDL terminology.

- **Fault handlers section**: This section is used to define the specific activities that occur as a result of a “fault” (error) resulting from invoking a Web service. BPEL4WS provides a uniform naming model for faults, requiring a qualified name, which is comprised of the target namespace of the WSDL document and the `ncname`\(^8\) of the fault.

- **Main processing section**: This section contains the description of the steps involved in executing the business process. In describing the business process steps, a number of key words are available for specifying particular activities, as indicated in the following table.

**BPEL4WS key words for activities**

<table>
<thead>
<tr>
<th>Activity key word</th>
<th>Activity description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;receive&gt;</code></td>
<td>Do a blocking wait for a matching message to arrive</td>
</tr>
<tr>
<td><code>&lt;reply&gt;</code></td>
<td>Send a message in reply to a message that was received</td>
</tr>
<tr>
<td><code>&lt;invoke&gt;</code></td>
<td>Invoke a one-way or request-response on a portType offered by a partner</td>
</tr>
<tr>
<td><code>&lt;assign&gt;</code></td>
<td>Update the values of containers with new data</td>
</tr>
<tr>
<td><code>&lt;throw&gt;</code></td>
<td>Generate a fault from inside the business process</td>
</tr>
<tr>
<td><code>&lt;terminate&gt;</code></td>
<td>Immediately terminate a business process</td>
</tr>
<tr>
<td><code>&lt;wait&gt;</code></td>
<td>Wait for a given period of time or until a certain time has passed</td>
</tr>
<tr>
<td><code>&lt;empty&gt;</code></td>
<td>Insert a “no-op” instruction into a business process (useful for synchronization of parallel activities)</td>
</tr>
<tr>
<td><code>&lt;sequence&gt;</code></td>
<td>Define a collection of activities to be performed sequentially in lexical order</td>
</tr>
<tr>
<td><code>&lt;switch&gt;</code></td>
<td>Select one branch of execution from among a set of choices</td>
</tr>
<tr>
<td><code>&lt;while&gt;</code></td>
<td>Indicate that an activity is to be repeated until a certain success criterion has been met</td>
</tr>
<tr>
<td><code>&lt;pick&gt;</code></td>
<td>Block and wait for the occurrence of a trigger; when the trigger occurs the associated activity is executed and the pick completes</td>
</tr>
<tr>
<td><code>&lt;flow&gt;</code></td>
<td>Specify one or more activities to be executed in parallel</td>
</tr>
<tr>
<td><code>&lt;scope&gt;</code></td>
<td>Define a nested activity with its own fault and compensation handlers</td>
</tr>
<tr>
<td><code>&lt;compensate&gt;</code></td>
<td>Invoke compensation on an inner scope that has already completed its execution normally</td>
</tr>
</tbody>
</table>

---

8 In XML terminology an “`ncname`” is a unique name without a colon.
Structure and Syntax of BPEL4WS Language

```xml
<process name="ncname" targetNamespace="uri"
    queryLanguage="anyURI"?
    expressionLanguage="anyURI"?
    suppressJoinFailure="yes/no"?
    enableInstanceCompensation="yes/no"?
    abstractProcess="yes/no"?

    <partners>?
        <!-- Note: At least one role must be specified. -->
        <partner name="ncname" serviceLinkType="qname"
            myRole="ncname"? partnerRole="ncname"?/>
        </partner>
    </partners>

    <containers>?
        <!-- Note: The message type may be indicated with the messageType
             attribute or with an inlined <wsdl:message> element within. -->
        <container name="ncname" messageType="qname">
            <wsdl:message name="ncname">...
        </container>
    </containers>

    <correlationSets>?
        <correlationSet name="ncname" properties="qname-list"/>
    </correlationSets>

    <faultHandlers>?
        <!-- Note: There must be at least one fault handler or default. -->
        <catch faultName="qname"? faultContainer="ncname">*
            activity
        </catch>
        <catchAll>?
            activity
        </catchAll>
    </faultHandlers>

    <compensationHandler>?
        activity
    </compensationHandler>

    activity
</process>
```
Appendix 2

Resources, Tools, and Utilities

- Microsoft Developer Network (MSDN) site for WSDL

- MSDN site for SOAP

- Apache SOAP http://xml.apache.org/soap/


- MSDN site for UDDI

- Business Process Execution Language for Web Services (BPEL4WS)


- Publicly available web services: http://www.xmethods.com