Communications of the Association for Information Systems

Volume 11 Article 30

April 2003

Web Services: Enabling Dynamic Business Networks

Bala Iyer
Boston University, bala@bu.edu

Jim Freedman
Boston University, jfreedma@bu.edu

Mark Gaynor
Boston University, mgaynor@bu.edu

George Wyner
Boston University, gwyner@bu.edu

Communications of the Association for Information Systems

WEB SERVICES: ENABLING DYNAMIC BUSINESS NETWORKS

BALA IYER
JIM FREEDMAN
MARK GAYNOR
GEORGE WYNER
School of Management
Boston University
BALA@BU.EDU

ABSTRACT

A Dynamic Business Network is a distinct system of participants (customers, suppliers, complimentors, competitors, service providers) that use the network to achieve customer satisfaction and profitability and where participants and relationships evolve over time. However, unpredictability and rapid change in a Dynamic Business Network creates a significant challenge in implementing and supporting business application software. Traditional information systems implementation methods require an a priori design and are built for a particular purpose for use over an extended period of time. Loosely coupled business networks change interrelationships between nodes both quickly and frequently, thus providing little or no notice for planning, implementing, or changing the supporting applications. The dynamic sourcing capabilities of the emerging Web Services framework provide a key to enabling these complex eco-systems. We explore the strategic and technological dimensions of Web Services and describe how they can be used to support dynamic business networks.

KEYWORDS:architecture, web services, dynamic business networks, modularity, stakeholders

I. INTRODUCTION

"Firms are embedded in networks of cooperative relationships that influence the flow of resources among them." [Gnyawali and Madhavan 2001].

The business landscape was once dominated by large hierarchical entities that are now being replaced by loosely interconnected organizational components. These firms face the choice of loose coupling or tight integration with all other entities involved in delivering goods or services. To understand these organizational configurations we take a network perspective. The network perspective of business is primarily concerned with inter-organizational relationships over time rather than with single exchange transactions [Nohria and Eccles 1992]. The unpredictability and pace of change in the relationships among nodes in the network make these systems extremely complex. Increasingly, the logic of general systems modularity is used to understand these networks [Langlois 1999; Schilling 2000]. Modularity provides a framework for understanding and managing complexity [Baldwin and Clark 2000]. Web Services, an emerging technology, provides a modular capability to combine, de-couple, and recombine software components to create virtual business applications in an ad hoc, real-time manner.

The main objectives of this paper are:

- to educate the reader about the concept of Web Services,
- how to use Web Services to support dynamic business networks and
- to understand the role of enterprise architecture in achieving that goal.

Most current discussions of Web Services overly focus on the developer/technical perspective without providing the business context. We provide a more comprehensive view by looking at Web Services from the perspective of different stakeholders – owner, architect, builder and end-user.

This article is organized as follows: In Section II, we describe a Web Service. In SectionIII, we introduce the stakeholder model. In SectionIV, we present the concept from the owner perspective. In SectionV, we discuss it from the designer perspective. The builder perspective, (SectionVI), presents a primer on Web Services. End-user related issues are discussed in Section VII. In Section VIII, we present conclusions and identify the limitations of Web Services.

II. WHAT ARE WEB SERVICES?

While the concept of Web Services was introduced several years ago, its definition is not agreed upon. For purposes of this article we use a specific meaning of Web Services proposed by an industry analyst:

Web Services refers to loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols" [Sleeper 2001].

Several key elements of this definition warrant further discussion.

Reusable Software Components

This concept is explained by the theory of modular design [Alexander 1964; Langlois 1999; Simon 1996].

Semantic Encapsulation of Discrete Functionality

Web services applets semantically encapsulate discrete functionality in the same way that objects encapsulate functionality in an object-oriented system. This notion of encapsulation is an important element in research on modularity. For example, Parnas discusses the advantages of designing a module "to reveal as little as possible about its inner workings" (Parnas 1972, p. 1056). Baldwin and Clark [1997] refer to Parnas' concept of information hiding as hidden design parameters. This concept of focusing on the specific needs of a particular application without regard to other functions is a key element of achieving synergistic specificity [Schilling 2000]. Encapsulating specific process knowledge within a discrete object is also a key element of sharing that capability within the framework of modular design.

ProgrammaticAccessibility

Web Services are programmatically accessible. Unlike web sites and desktop applications, Web Services are not designed exclusively for direct human interaction, and do not necessarily include a user interface. Rather, Web Services operate at the application level; they are called by and exchange data with other software. An example of such data exchange is in the dynamic processing of an order, where the line items are priced through one called module, and the tax is assessed through another called module. The inner workings of the program are transparent to the user of the system.

Standard Internet Protocols

Web Services are distributed over standard Internet protocols. They use the existing infrastructure such as hypertext transfer protocol (HTTP), file transfer protocol (FTP), simple mail transfer protocol (SMTP), and extensible markup language (XML), and conform to the standards and procedures adopted for using the Internet. This element is important for the successful adoption of Web Services. The success of the Internet is largely due to the simplicity and flexibility of the layered architecture of the technology that supports the packaging and transport of data and provides end-to-end services. Web Services are designed to use that packaging and transportation mechanism already adopted by firms worldwide. Web Services become yet another type of traffic on the Internet similar to the World Wide Web, e-mail, or voice over IP traffic.

Both a technical and a conceptual meaning are associated with Web Services. From a technical point of view, Web Services are a layered set of standards and protocols similar in nature to the layered set of standards and protocols that support the Internet. However, the relationship between the layers, in this case, is not a stack but a network as illustrated in Figures 1 and 2. The abbreviations in Figure 2 are explained in the text that follows the figure.

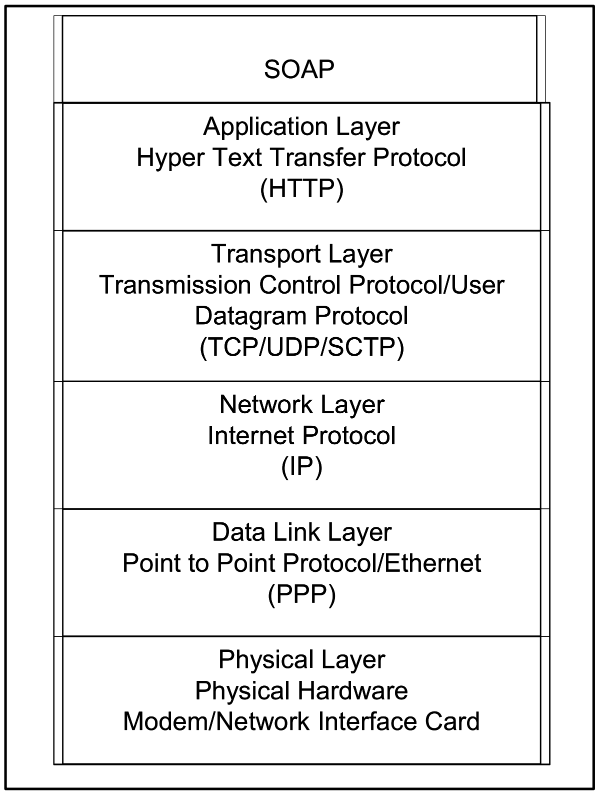

Figure 1. Internet Layered Stack

Web Services provide a standard way for heterogeneous systems to share and exchange information. Web Services are not a new idea, but the continued evolution of many older ideas.

What is new is that Web Services are based on generally accepted open standards. These standards include:

- Simple object access protocol (SOAP) for the message structure,
- Extensible markup language (XML) for data encoding,
- Web services description language (WSDL) to describe the application programming interface (API) detailing the interface explaining how to use the service, and
- Universal description, discovery, and integration (UDDI) to register a service so others can discover it.

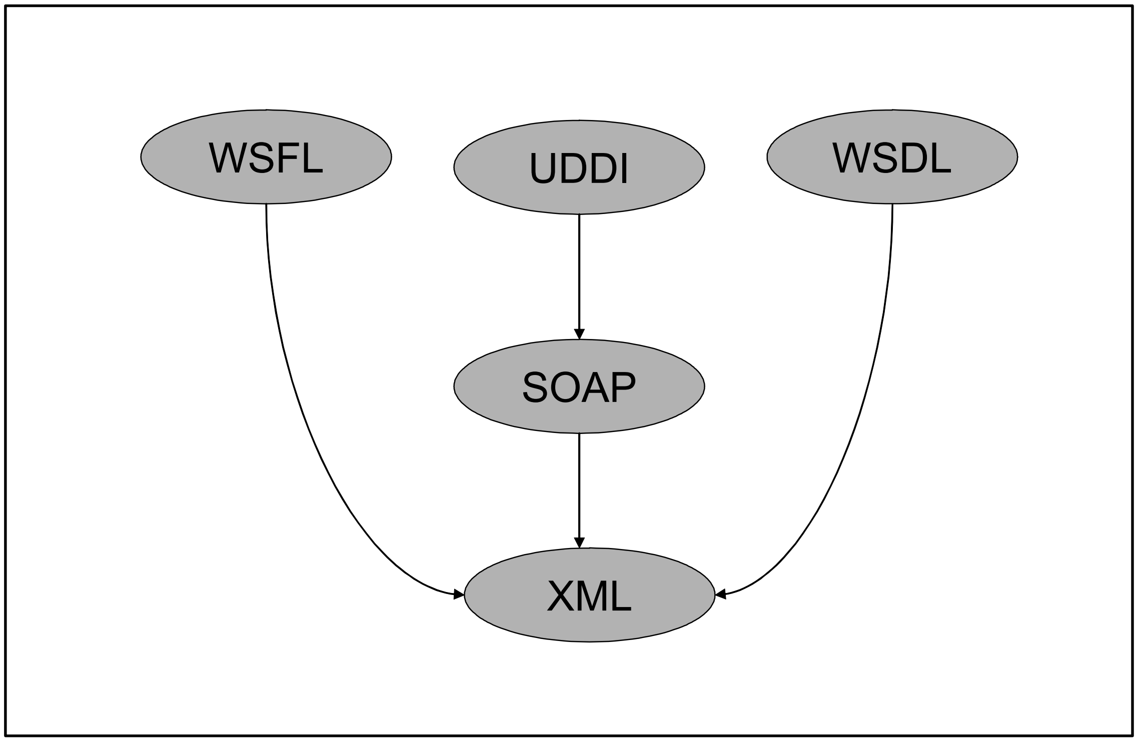

Figure 2. Web Network Services

At the highest level, a web service is a resource on a computer invoked by sending it a message using the SOAP protocol. This SOAP message contains the information needed to perform the web service. For example, Figure 3 includes a simple web service to report the Manufacturer's Suggested Retail Price (MSRP) for an item. This pricing service is listed in a directory of Web Services along with directions for its use. The client can look up the web service and call it to find the price of a given item. Both the request and response are sent in a SOAP message. This information is encoded in a standard way: XML. Since all Web Services agree to use XML, all clients can access Web Services as long as they follow the standards. This arrangement is very similar to how a web browser allows users access to web applications that follow the HTML standards. Web Services are not a new idea, but rather, the next generation of Remote Procedure Calls (RPC) [Birrell and Nelson 1984; Orfali et al. 1996]. Web Services promise the Holy Grail for IT professionals: a standard way for linking any two systems so they can exchange information.

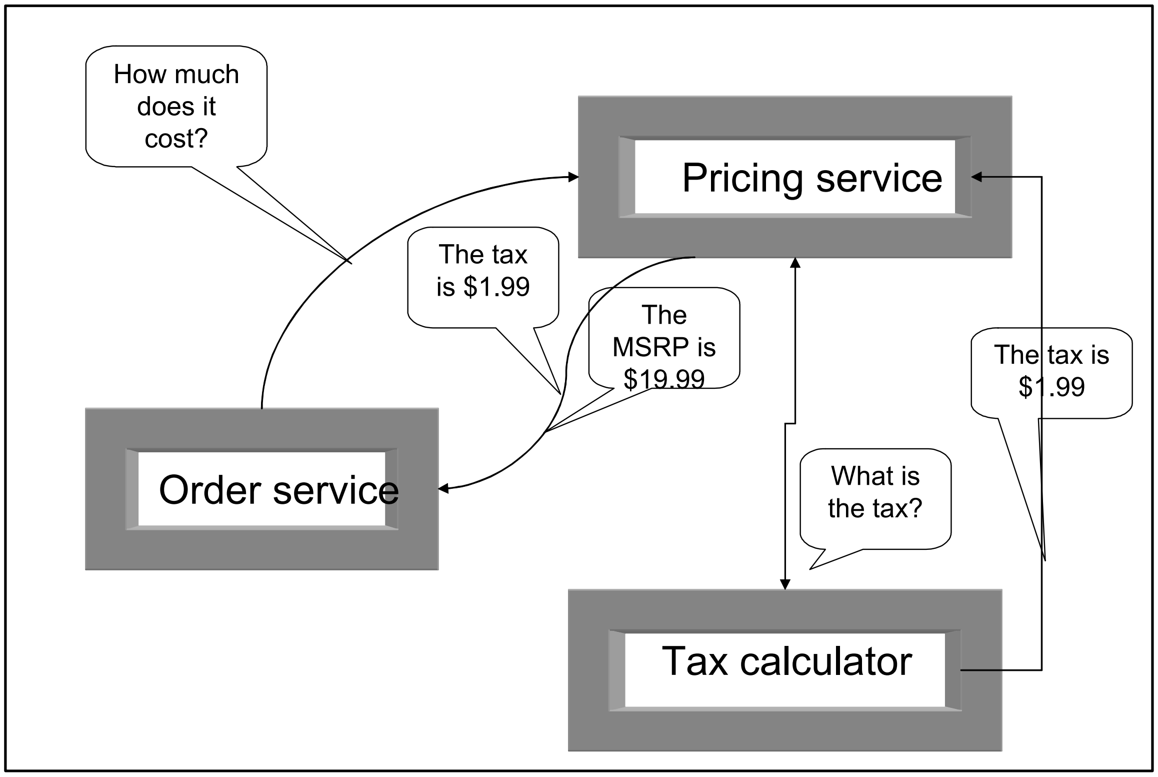

Figure3. Web Services Example

III. STAKEHOLDER MODEL FOR WEB SERVICES

To define, design, develop, and deploy Web Services applications, we first identify four different stakeholders (owner, architect, builder, and end-user) and questions they should ask [Zachman 1987]. Figure 4 shows the views of each of these stakeholders.

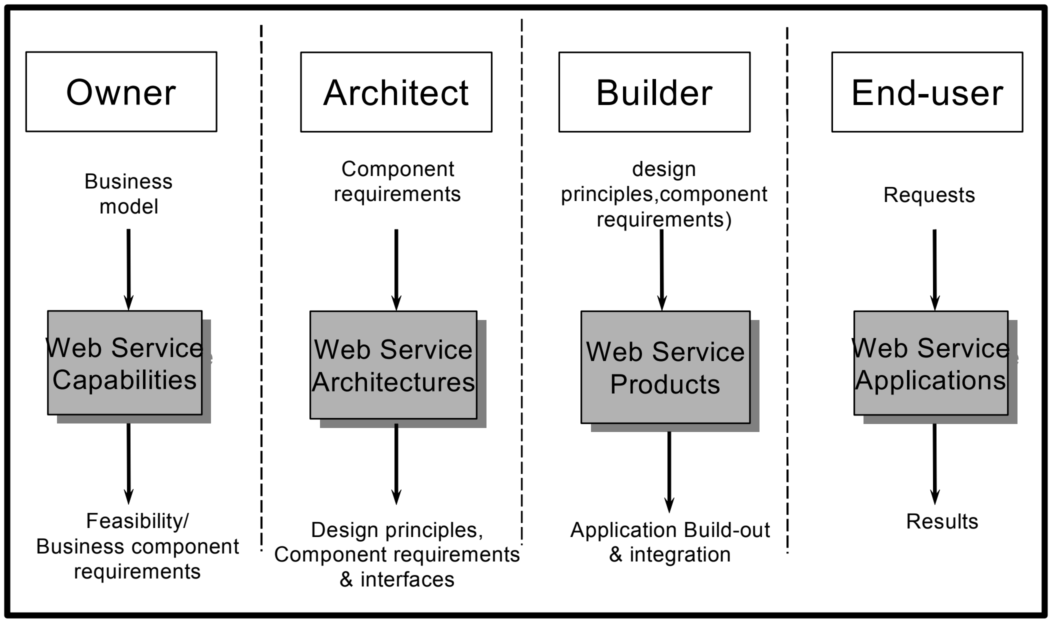
Owner
Business model
Web Service Capabilities
Feasibility/ Business component requirements
Architect
Component requirements
Web Service Architectures
Design principles, Component requirements & interfaces
Builder
design principles,component requirements)
Web Service Products
Application Build-out & integration
End-user
Requests
Web Service Applications
Results

Figure 4. Stakeholder Views of Web Services

OWNER

This person must analyze the Web Services capabilities necessary to implement a business model that they identified. A business model is a method by which a firm builds and uses its resources to offer its customers better value than its competitors and to make money doing so. The business model details a set of specific activities that a firm should perform or source from a provider. These activities can be specified as a set of business components/ services. The major assumption is that if these activities are performed well, the firm will make money. The owner asks: “How can Web Services enable my business model?” The answer takes the form of a set of business component requirements with associated budget allocations and performance metrics.

ARCHITECT

This person is responsible for defining the design principles, individual components and the interfaces between them. The architect understands and uses the component requirements, Web Services frameworks, and system design theories to create a set of architecture design principles. The architect asks:

- “What principles and commitments will guide the design of the required components?”
- “Which of those business requirements could we provide using Web Services?”
- “What are the advantages/disadvantages of delivering the requirement using a Web Service?”
- “Which components should be built and what functionality should they encapsulate?”

BUILDER

A builder implements and integrates components using the architecture design principles as defined by the architect. To do so, the builder should identify registries that list the required components, search for the components/services, and learn how to use them (inputs and outputs) according to their advertised performance metrics. The builder asks:

- “What vendor provided solutions will I deploy to implement and integrate?
- “What granularity of components should I select?”
- “How long would it take, and what would it cost to develop each Web Service?”
- “What new skills would we need to acquire to develop these Web Services?”
- “What products/tools/technology should we purchase?”

END-USER.

This person is mainly interested in doing his or her work. The end-user makes requests to applications and expects reasonable responses. The end-user asks:

- “How do I use this set of services to perform my business responsibilities or solve business problems?”
- “What could I do with the new application that is currently difficult for me to do?”

In the next sections we present each perspective in greater detail.

IV. OWNER’S PERSPECTIVE

In many interconnected markets, business is changing rapidly. Firms face increasingly dynamic market demands. The Internet increased both the speed of communication and geographic reach. Firms that previously focused locally can now compete economically on a global basis. The changes that impacted the personal computer industry exemplify the effect of dynamic markets across organizational boundaries. The relationships among the PC manufacturer, component suppliers, transportation firms, and the end customer change continually. When buying a PC it is now expected that the specific configuration can be customized to any particular need with ease. Visibility of the order is open to all of the stakeholders concerned with the delivery of the end product to the customer.

The combination of rapidly changing market needs and new entrants in the marketplace make the future more uncertain than ever before. At the same time, enterprising firms are beginning to change their business models to manage the unknown. Flexible, "loosely coupled" network organizational models are starting to replace rigid, "tightly coupled", integrated organizational models. This evolving strategy lowers the risk associated with a rapidly changing market by modularizing the research, marketing, production, delivery and support/service processes. Web Services allow information systems to support the flexible designs that a Dynamic Business Network environment demands. In this environment, the interface requirements often are not known a priori. Using Web Services architecture, applications may interface dynamically, sharing data that was not determined in advance.

Maintaining competitive advantage in a dynamic marketplace continually demands new and different capabilities, competencies, and resources. As the marketplace's needs shift so do the business models of marketplace leaders. Flexibility is needed to adjust quickly to changing marketplace conditions, new competitive offerings, changes in supplier resources, and new product design requirements. Dynamic business networks that easily and quickly change to meet the ever-changing demands of the marketplace reap the greatest rewards in this unpredictable environment. However, the advantages gained by flexibility also come with a cost in governance mechanisms and performance (scalability, velocity of throughput), trade-offs that may be counter productive in more traditional marketplaces. While dynamic marketplaces reward flexibility, other highly regulated or less dynamic "traditional" marketplaces reward integration and systems with a high degree of "synergistic specificity" [Schilling 2000]. This more traditional view [Chandler 1964] of the marketplace measures market leadership by reduced cost of production and increased quality. Marketplace leaders are often represented as monolithic integrated production capabilities provided by a single firm. The flexibility provided by modularity provides little advantage to these tightly coupled systems.

INFORMATION SYSTEM SOURCING

The owner's perspective results in the selection of a sourcing strategy – build, buy, rent, or share – for software. Appendix II describes the evolution of software development practices and the rise of outsourcing.

A new paradigm is emerging where dynamic business networks evolve over short periods of time. The boundaries between supplier, customer, competitor, and business partner are blurred and dynamic. In one business situation two organizations may be competitors, while at the same time the opportunity presented by another business situation leads these two organizations to behave as partners. New "partnership" business relationships require the organizations to share information in order to work closely. Systems designed to provide internal information may need to share that information with other, unknown systems quickly.

For example, FEDEX, a business that specializes in the transportation of goods, may need to share specific information about the location and status of a particular package sent by one of their customers (XYZ Computers) with a third party (in this case XYZ Computer's customer) farther down the value chain. The package tracking information needs to be available seamlessly on the XYZ Computer website. Since XYZ is in the business of selling computers, they may want to provide their customers a variety of shipping options and may offer alternatives to FEDEX, such as UPS. XYZ Computer therefore wants to provide the same access to shipping status information for any vendor that ships the product to XYZ's customer. This ability to link different information dynamically from different organizations is an example of the flexibility that is expected in the dynamic business network.

SIDEBAR I. DYNAMIC BUSINESS NETWORKS

The term business network is being used to describe many contemporary organizations in the computer and biotech industries. It is argued that to be successful, organizations need to be part of *dynamic business networks* that adapt and respond to changing business and technological conditions. Business networks are characterized by lateral and horizontal linkages within and among firms [Nohria and Eccles 1992]. Several premises underlie a network perspective of organizations [Nohria and Eccles 1992]:

- All organizations are social networks in important respects and need to be addressed and analyzed as such.
- An organization's environment is properly seen as a network of other organizations.
- The actions of actors in an organization can be explained in terms of their position in networks of relationships.
- Networks constrain actions, and in turn are shaped by them.
- The comparative analysis of organizations must take into account their network characteristics.

Based on previous published work [Bovet and Martha 2000; Brandenburger and Nalebuff 1996; Tapscott et al. 2000], we define a business network as a distinct system of participants – (*customers, suppliers, complementors, competitors, commerce service providers, and infrastructure providers*) that use a network to achieve superior customer satisfaction and profitability. The business network is a clear value proposition. It is also a business platform that defines the standards and rules for exchanging goods and services within the network.

- A *customer* not only receives value but also contributes to the business network.
- A *commerce service provider* enables the flow of business, including transactions and management, security and privacy, information and knowledge management, logistics and delivery, and regulatory services (Exchanges, SEC).
- *Suppliers* supply the primary inputs that go into the manufacture of the product or the service.
- A *complementor* is a firm in the network that makes the product/service produced by the network more valuable to the customer than if the product/service were offered alone (e.g., Intel and Microsoft).
- *Competitors* are firms in the network that reduce the value of the product provided by the network (e.g., Coca-Cola and Pepsi-Cola).
- *Infrastructure* providers deliver communications and computing, buildings, offices and similar underlying services (e.g., DIGEX, Verizon).

We consider these networks *dynamic* because the participants and their relationships evolve over time. For example, in the case of a supply chain management system, the e some of the changes that may occur include: number of items produced, the type of item produced, the partners that produce these items, and the standards used to exchange information. The linkages between the entities are typically information based and are implemented using information systems.

Traditional applications are not designed to disaggregate and reaggregate. Rather, they are built focused on a particular purpose and must be modified when the business requirements change. The new generation of information systems must be created ad hoc to meet the particular requirements created by the convergence of entities in the dynamic business network. This method of sourcing information systems is what we refer to as "Dynamic Sourcing". Web Services, a type of dynamic sourcing, is specifically designed to provide more flexibility to support unknown business requirements and business organizational structures. It also provides a new option for choosing applications: sharing applications on a case-by-case basis to meet a particular purpose.

V. DESIGNER'S PERSPECTIVE

Although the promise of dynamic business networks is compelling, to participate in such a network a firm must leverage its internal resources. Many multi-divisional firms find it difficult to manage disparate resources. Clark and Peruzzi [2002] found that, Web Services are seen as a way to

1. integrate incompatible computer systems, and
2. share data, applications, and business process internally.

Their survey presented responses from 796 people in 50 enterprises with over $10 million in annual revenue. The responses suggest that early adopters of Web Services

> *"get new ways to integrate applications internally. Integration helps unlock the value of all their previous software installations by letting them leverage old systems as they implement new ones. Connectivity creates value, and Web Services provide a new, cheaper than EAI tool to link applications."* [Clark and Peruzzi, 2002]

The business challenge addressed by enterprise-wide integration is the need to leverage organizational capabilities, streamline internal processes and create a common, simplified interface with external stakeholders. Many enterprises grew through mergers and acquisitions. Historically, management would use financial measurements to judge each operating unit based upon their individual performance. This approach of "each ship on it's own bottom" provided little incentive to consolidate systems or processes across the organization. With increasing use of the Internet to communicate with external stakeholder (customers, suppliers, partners, or investors) firms are under increasing pressure to simplify their interfaces. Customers want to use one portal to access any product or service rather than use different methods to inquire into what the firm can provide. Stovepipe systems preclude doing business in this way. A similar business value can be derived in a multi-divisional enterprise by consolidating purchasing capacity and leveraging relationships with vendors. Systems built to function separately require custom-designed interfaces or need to be replaced by enterprise resource planning (ERP) systems. Web Services promise to provide the required integration capability in a simple, inexpensive manner and thereby become an attractive alternative to the more traditional means.

We next turn to modularity, an important characteristic that is central to the value that Web Services provides.

MODULARITY AND VALUE

Many natural (organisms and ecosystems) and human constructed (mechanical, intellectual, organizational and social systems) systems that we encounter on a daily basis are complex. A complex system is

> *"one made up of a large number of parts that interact in non-simple ways. In such systems, the whole is more than the sum of the parts, at least in the important pragmatic sense that, given the properties of the parts and the laws of their interaction, it is not a trivial matter to infer the properties of the whole."* [Simon, 1966]

Modularity, a general systems concept, is proposed as a way to handle and understand complex systems. Schilling [2000] refers to modularity as a continuum that describes degree to which a system's components may be separated and recombined. Modularity refers both to the tightness of coupling between components, and the degree to which the "rules" of the system architecture enable (or prohibit) the mixing and matching of components. Langlois [1999] relates the value of modularity to the need to understand and manage complex or dynamic systems. He refers to the constructs of visible design rules as *modularization* and defines the constructs as follows:

- An *Architecture* specifies what modules will be part of the system and what their functions will be
- *Interfaces* describe in detail how the modules will interact, including how they fit together and communicate
- *Standards* test a module's conformity to design rules and measure the module's performance relative to other modules.

Hidden design parameters are decisions that do not affect the design beyond the local module. Hidden elements can be chosen late and do not have to be communicated to anyone beyond the module design team.

In a stable, predictable environment, great advantage is derived from tightly coupling components into a highly integrated system. This focus on streamlining systems is the essence of the industrial revolution where competitive advantage was derived from lowering the cost of production through designing and managing very narrowly focused production capability. In contrast, the value that is derived from modularity lies in decomposing systems to cope with a changing environment. "Modularity is a very general set of principles for managing complexity." [Langlois 1999] If the environment is static, and the system functions as expected, then there would be no need to understand how it is constructed.

Software components that are designed to be loosely coupled, such as in Web Services allow for much more flexibility than traditional methods for connecting components. The traditional approach is to define the interfaces between software components clearly and to develop specific Application Program Interfaces (API) to transfer information between the components. The general notion is that the more closely coupled the components, the more efficient the application. At the same time, the more tightly coupled the application the more inflexible is the code to changes in structure. A good example of the inflexibility of tight coupling was seen in the Year 2000 software problem.

The theory of modularity provides the enterprise a broad set of principles than can guide the design of the software components within a system. These components, like 'LEGO' blocks, provide the building blocks and the flexibility to mix and match components to meet unknown future systems requirements. Web

Services, on the other hand, provides the technology to specify the components that were designed using modularity theory, the interface to each component, and the ability to dynamically mix and match components to implement future systems. In addition to this, Web Services allow the enterprise to build custom 'LEGO' blocks to meet unique needs.

How does an enterprise put a value on modularity? Baldwin and Clark [2000] apply real option theory to study modularization in the computer industry. They show how modularization in computer systems (like the IBM 360) changed the industry. Modularized computers consist of components that define interfaces. Because each component conforms to its interface rules, modules that follow the defined interface are interchangeable. In contrast, an interconnected system cannot swap components because only a single massive component exists. Baldwin and Clark's work shows how modularity increases value, and how increasing technological uncertainty about the value of the modules increases this value of modularity.

Appendix III presents an example of how modular design provides value.

ENTERPRISE ARCHITECTURE FOR WEB SERVICES

In the previous section, we discussed the concept of modularity and its impact on an enterprise's flexibility. Web Services is the technology that can be used to design for this flexibility. However, neither modularity theory nor the Web Services technology provides the designer with any guidance on the list of things or entities that can be considered for modularization. In this section, we present one such list of entities that we call enterprise architecture, based on the types of knowledge or logic that an enterprise typically captures, and upon which we can apply the principles of modularity.

We begin to identify the list of entities or enterprise architecture by considering a business process. A business process is a complete coordinated thread of all the serial and parallel activities needed to deliver value to the enterprise's customers [BPMI 2002]. Traditionally, enterprises used software applications to support coordination and execution of business processes. Although these applications were reliable, they were not built to be very flexible, agile, or transparent, because they combined and tightly couple the various assumptions about processes, data, how the business functions and how the applications are designed. This makes it very difficult to locate and make changes to any one of the assumptions. Recently, due to the availability of technologies such as components and Web Services, enterprises are investigating approaches to modeling business processes using more flexible application software development strategies. One such approach -- the modular, object orientedapproach being investigated is compared to "LEGO blocks," combining, disassembling and recombining basic business functions to create specific business processes to satisfy different business objectives. This approach uses business processes to satisfy different business objectives. This approach has resulted in tools that help separate a business application into four domains: process logic, application logic, business rules and data (Figure 5).

Enterprise Process Logic Layer. This layer defines a standard method to exchange business messages, conduct trading relationships, and define and register business processes.

Application Logic Layer. This layer defines the presentation logic, business engines and the integration logic for an enterprise application. The presentation logic deals with client requests by handling them directly or using a broker to deal with them, then sending the responses back to the user. Business engines are the set of application services, from forecasting to credit card processing, that automate particular business functions. Some of these applications are proprietary and confidential to a particular company or group of allies, while others are public and can be shared with chosen partners. In some cases, companies may develop their own application services and then choose to sell them on a subscription basis to other enterprises, creating new and potentially lucrative sources of revenue [Hagel III and Brown 2001].

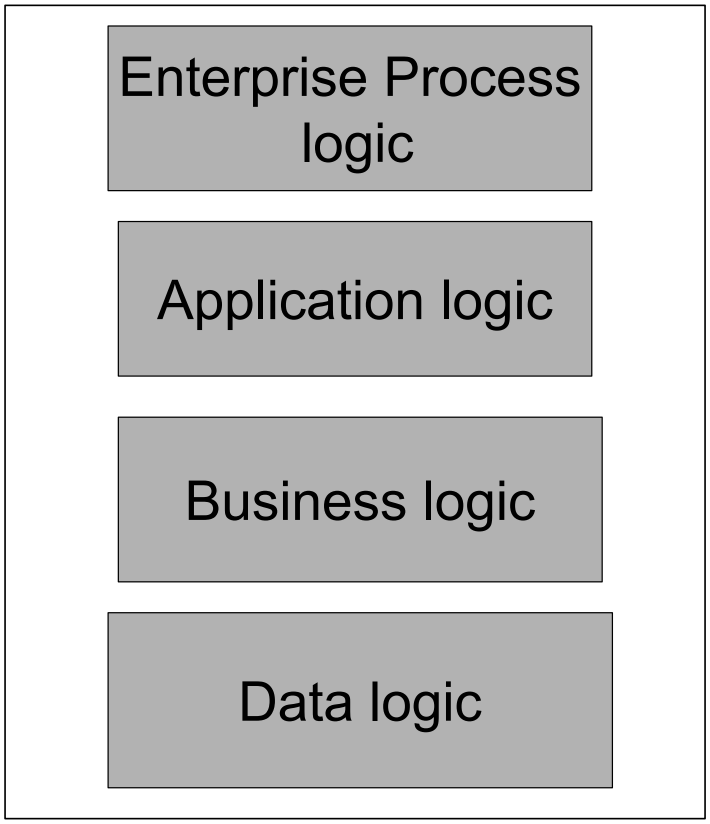

FIGURE 5. The Enterprise Architecture

Business Logic. The next layer contains the workflow logic and the business rules. The Workflow Management Coalition defines workflow *as*

> *the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules* [WFMC 2002].

Modern enterprises store the workflow logic as business rules [Morgan 2002].

Data Layer. The data layer, shown in Figure 5 includes the metadata about the information objects in each of the enterprise process logic, application logic, business rules, and data sub-layers. It also includes the metadata that represents the inter-relationships and associations between the information objects in each of these sub-layers. The last category of data in this layer is the metadata repository that includes the following: metadata about the business data captured in the system such as its source (person, role, business unit, location, and associated process that captures it), date/time of capture, granularity (hourly, daily, etc.), and its storage location in the system (which database/file, specific table(s), whether duplicated in case of distributed data, etc.). A corresponding set of metadata that describes the models (role/person/business unit responsible for maintaining it, storage location in the system, date/time of creation/revision) as well the model components is also part of the metadata repository.

The logic behind each layer presented in Figure 5 can be captured in a standardized form using XML. Furthermore, using the directory services provided by Web Services, this information can be accessed by other applications within or outside the enterprise.

Dynamic business networks are held together by a set of information standards, which function as the *lingua franca*, enabling network participants to exchange information about such factors as business rules, customers, products, and applications. By separating the enterprise logic into the layers defined in Figure 5, organizations can easily establish dynamic business networks with other organizations by exposing the logic layers selectively. This approach enables a company to "orchestrate" critical cross-company processes as though they are part of the company, while specializing in their internal competencies.

For such networks to function smoothly, several conditions must be met [Hacki and Lighton 2001].

1. Business standards must emerge and be accepted in every layer – enterprise process logic, application logic, business logic, and data logic. For example, to automate connections between organizations, shared meaning for terms such as prices (per pound, per kilo, etc.) and quality must be defined.
2. Rigorous performance standards must be met.
3. Sharing of benefits generated across all partners should be equitable.
4. All key business processes – (project management, order entry, HR administration, and budgeting) should be online.
5. Development and testing of new opportunities within network partners should be online.

WHY ARE WEB SERVICES BETTER?

Previous generations of distributed computation environments did not display the flexibility that Web Services do. Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), and Remote Method Invocation (RMI) are based on the RPC paradigm with tight coupling between what the client sends and what the server expects. The type and order of passed parameters are rigorously enforced because the parameters are marshaled and un-marshaled for the Web Service. This coupling is tighter than what is required with Web Services because Web Services allow both a RPC and message paradigm. The RPC style of Web Services is a mapping of the RPC paradigm: place the XML encoding of the parameters into a SOAP envelope. The message passing model is far more flexible because of its looser coupling between client and server.

Many big vendors in the pre-web service days are leading the push to Web Services. A vendor's Web Services development environment is aligned with their history regarding other distributed environments they supported. Sun's RMI and J2EE (Java 2 Platform, Enterprise Edition) are both tightly coupled to Java. Microsoft's DCOM favored the Windows platform, while their .NET requires it. The design goals of CORBA and .NET are similar, but split the market so that big Unix vendors, open source, and Microsoft never agreed about what the common language should be between heterogeneous computers.

Web Services emerged as the best choice from the many different experiments in distributing computing environments. For the first time, industry agrees to a common method to access remote resources across heterogeneous networks and systems. This agreement is a good example of how learning from many generations eventually leads to a solution that is acceptable to most of the vendors – not an easy feat.

One powerful attribute of Web Services that encourages innovation is the loose coupling between clients and servers. By agreeing on standards such as SOAP and XML, transferring data between all heterogeneous systems becomes easy. These standards explain how typed data is exchanged between systems that disagree about how data is represented. This loose coupling implies easy replacement of one web service with another, provided the defined interfaces of both Web Services are identical. This loose coupling between the client and server with Web Services gives consumers and developers flexibility in building and evolving these services because it promotes experimentation.

The vendors agree on the big picture of Web Services, however, they have different ideas about how Web Services should be implemented. The biggest players have business models that play to their strategic business advantages: Microsoft believes in using the Windows platform, while Sun is focused on the Java language. Some environments are less restrictive. such as Axis from the Apache group [Apache 2002]. Each system offers advantages and disadvantages – the best choice depends on the particular attributes of the individual organization, and what it wants to do. Fortunately for users, the value of Web Services is independent of how they are built. The major vendors agree about what counts the most – it's not how you build Web Services, but rather, it iss what users can do with these services that creates the most value for everyone.

The architect's perspective results in the determination of the modularity for the enterprise logic. The choice includes data, applications, business and enterprise logic. The architect can chose to modularize one, several, or all of these logics.

VI. BUILDER'S PERSPECTIVE

Web Services provide a deployment framework for efficient development and interoperability of information systems using industry standards.

Builders suggest that by using modular design concepts and a Web Service-oriented architecture, applications can be assembled, disassembled and reassembled more easily, i.e., maintained. A major theoretical advantage of using standards such as XML [XML 2002] and SOAP [Graham et al. 2002] is that the original software application does not need to be rewritten to be shared. Given the huge number of applications that are currently in production worldwide, the idea of "wrapping" a functioning application using a standard language designed specifically to share data and processes between computer systems sounds like the "silver bullet" that technology innovators sought since the beginning of computing.. The concept of Web Services provides a vehicle for applications to both describe their function and data but to also interface with other applications dynamically.

INFORMATION SYSTEM INTEGRATION

Sharing information across the enterprise or between enterprises requires the detailed analysis and design of application program interfaces or APIs. The three broad approaches to systems integration are [Markus 2000; TechMetrix 2002]:

1. *Data warehousing (DW)*. In this approach (Figure 6), an organization leaves its "source" systems (those that need to be integrated) alone. Instead, extracts are taken from these systems and loaded into a "warehouse" from which sophisticated analysis can be done using analytical tools.

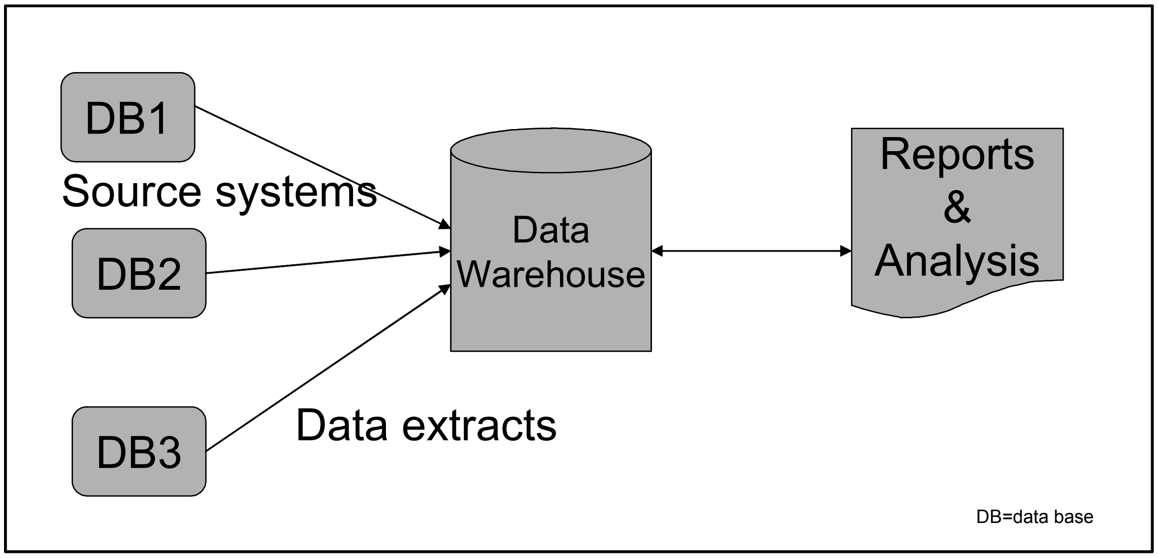

Figure 6. Data Warehousing

2. *Point-to-point integration (P-to-P)* (Figure7), involves developing a unique, customized integration solution that can link existing applications. These solutions are generally based on a client/server structure, and on various communication middleware applications. The client/server concept facilitates the distribution of functions to the application server that performs the task. This approach is inexpensive and is implemented quickly. The most significant trade-off is the ever-increasing number of point-to-point connections that must be created and maintained when a new system needs to be integrated. In the case of the example shown in Figure 6, the addition of a sixth component would potentially require the creation of five point-to-point connections. This approach results in a higher risk of error and increased system lag times.

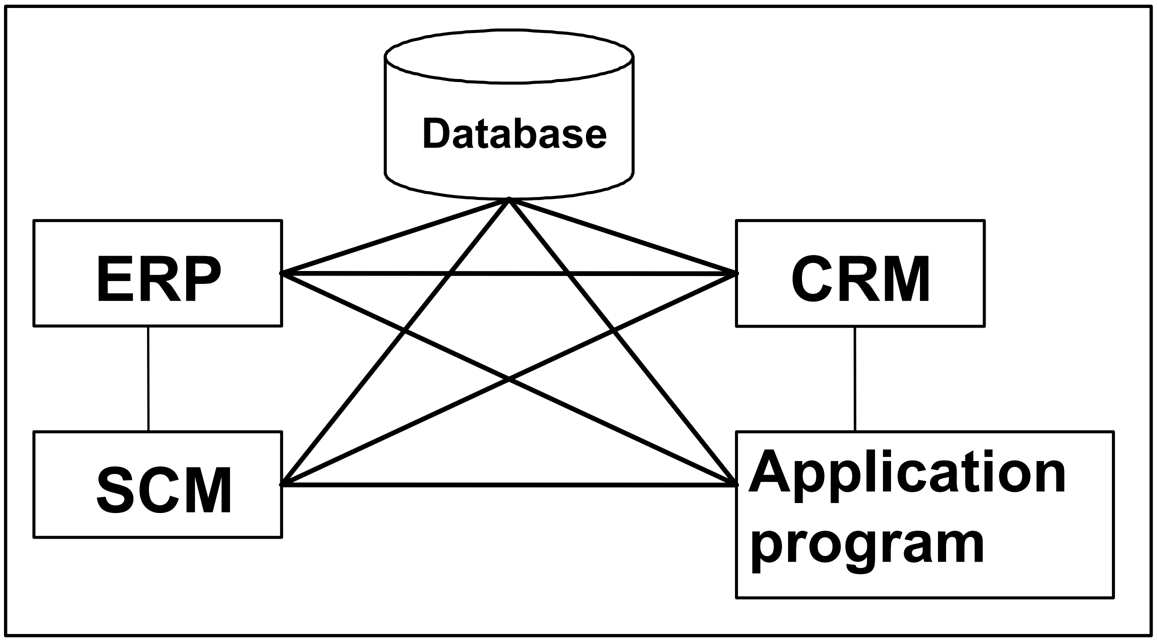

Figure 7 Point-to-Point Integration

3. *Enterprise Application Integration* (EAI) [Linthicum 2000], uses a central negotiator which manages the interaction among applications (Figure8). The EAI provides a single common connection between the applications.

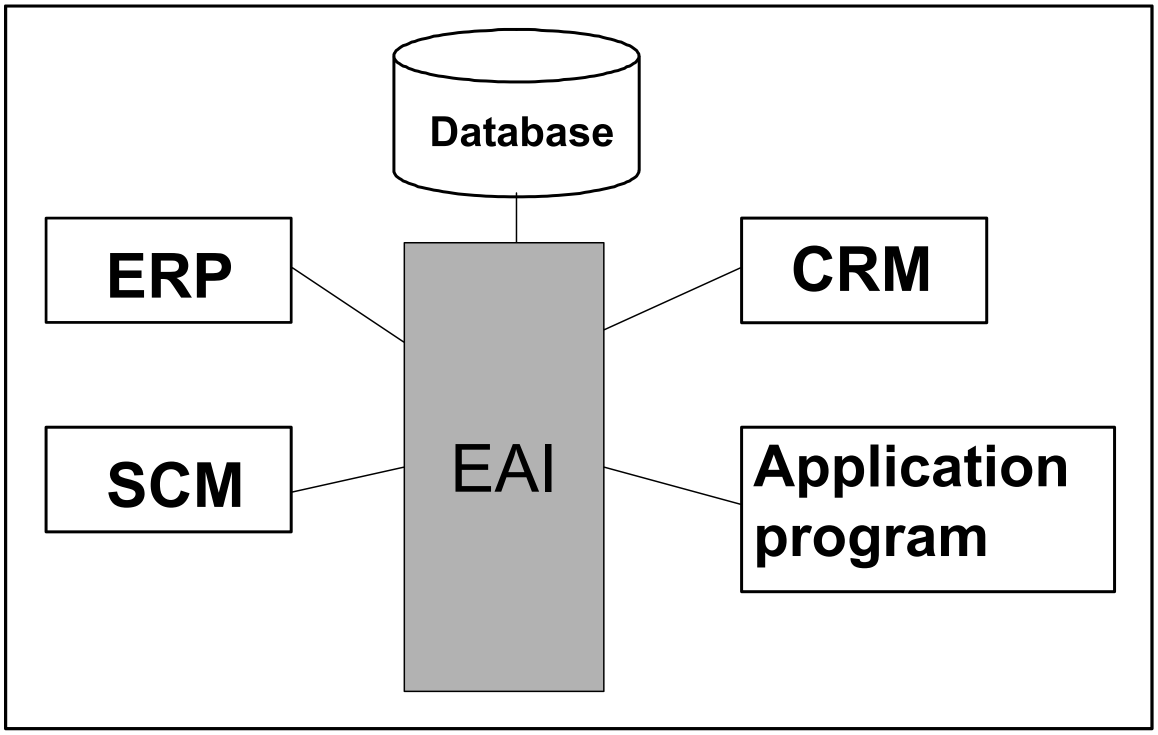

Figure 8. Enterprise Application Integration (EAI) Approach

Using an intermediate communication format allows all integration to be accomplished at a single node. Adding an additional application requires only that a single point-to- point connection be created, between the application and the EAI node. These tools provide useful features such as:

- Ready to use adaptors for connecting common applications to the central node,
- Advanced message routing between applications,
- Message transformation,
- Management of complex inter-application processes, and
- Administration of flows and processes.

With the increasing need for customer fulfillment, real time management and lean management, some organizations want all the components across the business network able to communicate in real time and to synchronize with one another. E-business often requires that an enterprise's processes be able to integrate extensively both with those of its partners and with complementary applications. Meeting this requirement is no longer a matter of simply connecting databases, and requires an integration platform that communicates with all the applications in the information system.

4. These needs led to the development of a fourth approach to System Integration Architecture, developing and implementing a *Network Platform* (NP). This platform expands on the EAI approach by adding the ability to share data, applications, process, and business rules to launch many new applications. The Network Platform (Figure 9)

divides the data, process, applications, and business rules elements into their logical and operational layers. The logical layer contains the enterprise logic (as described in section 5) and the operational layer contains the actual components or services that implement the required functionality. to customize the services for the various market segments, the enterprise logic platform includes a set of APIs that can be called by the application programs to customize the services for the various market segments, the enterprise logic platform has a set of APIs that can be called by the application programs. These APIs can help launch products for various segments or to meet a variety of needs within a segment. Similarly, the operating platform provides a set of APIs that will provide each enterprise with the option to use services provided by various business units from other enterprises. Web Services is one form of Network Platform.

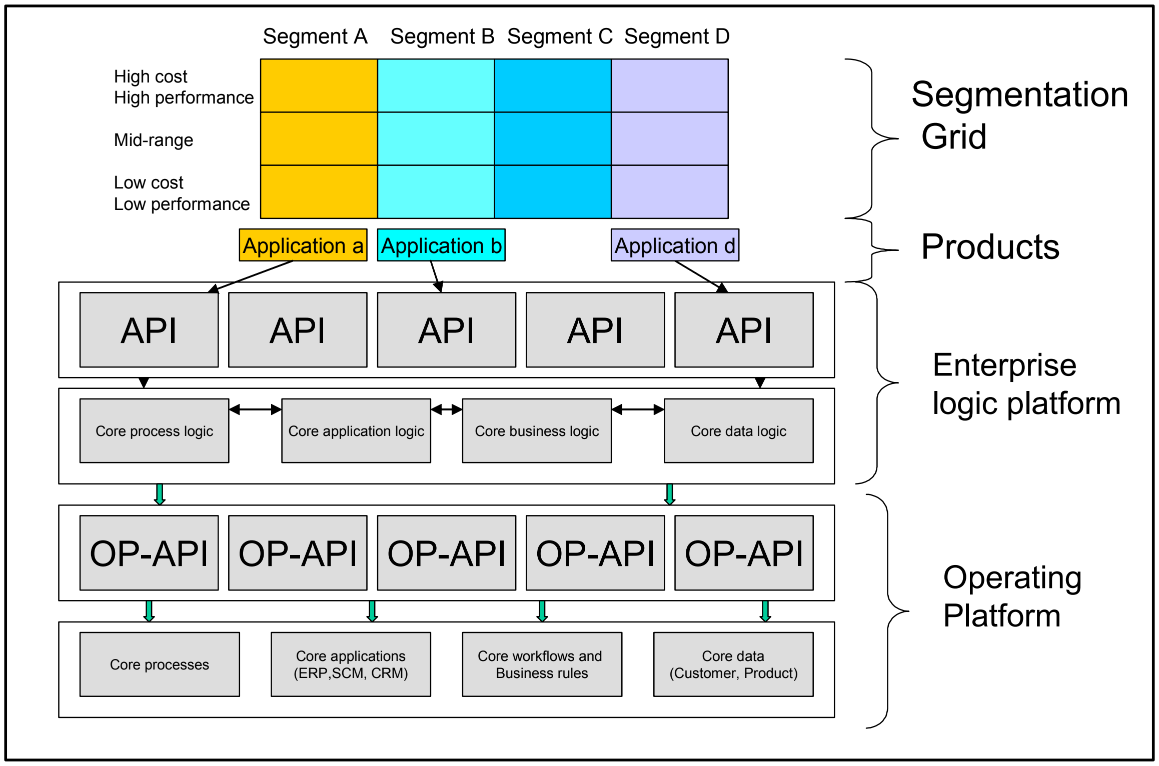

Figure 9. Network Platform (NP) Approach

To summarize, four options are available to source applications; Rent, Buy, Build or Share (Dynamically Source)[McKeen et al. 2002]. To integrate applications, four options: data warehousing, point-to-point, enterprise application integration and network technology platform can be used. To summarize, four options are available to source applications; Rent, Buy, Build or Share (Dynamically Source)[McKeen et al. 2002]. To integrate applications, four options: data warehousing, point-to-point, enterprise application integration and network technology platform can be used.

VII. END USER'S PERSPECTIVE

In the previous sections we defined what a web service is, and described how to use one. When you are ready to make a Web Service available, you can publish its characteristics by providing its description in an XML document using WSDL. To make it easy for developers and applications to locate it, you can place these WSDL descriptions in a private or public UDDI registry[1].

[1] The acronyms are defined in Section one and in the List of Acronyms at the end of this article.

A developer could “call” a Web Service by using a URL tag and WSDL description of a particular service to use. They can find this information by querying a UDDI registry.

When a Web Service is ready to “call” another service, it sends a request as an XML document in a SOAP envelope. This protocol can work across a variety of transport mechanism, either synchronously or asynchronously.

Figures 10 and 11 illustrate how these pieces fit together.

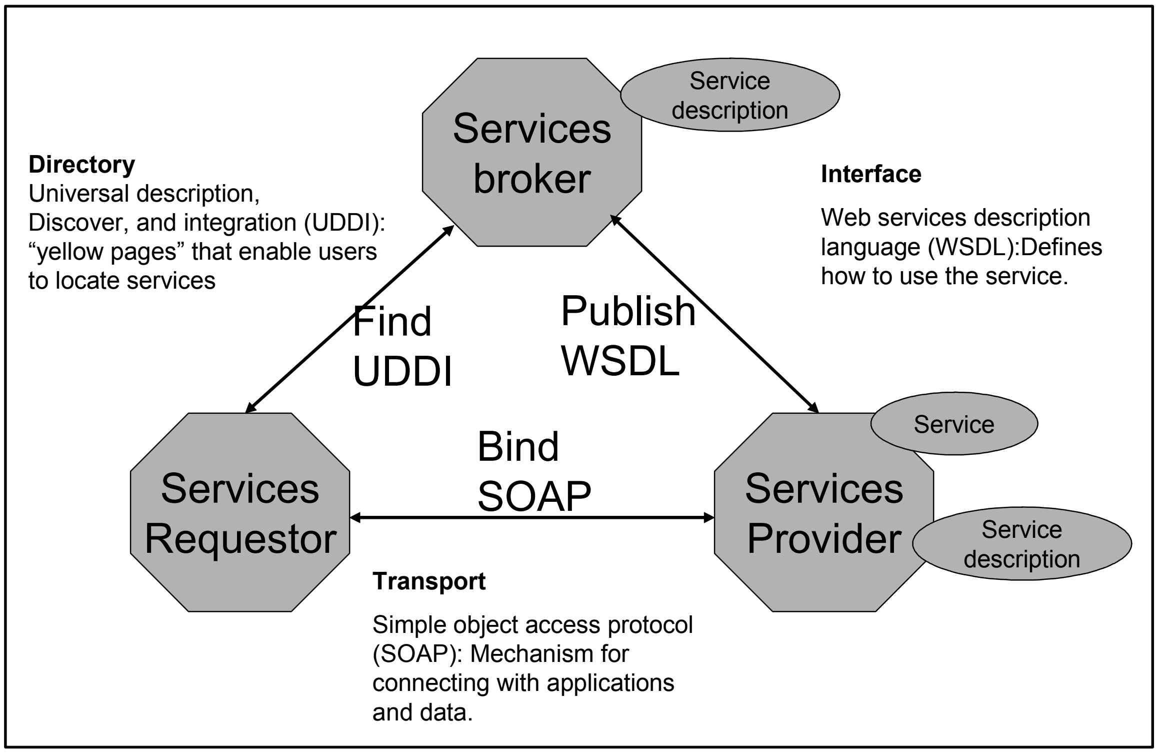
Services broker
Service description
Directory
Universal description, Discover, and integration (UDDI): “yellow pages” that enable users to locate services
Interface
Web services description language (WSDL):Defines how to use the service.
Find UDDI
Publish WSDL
Bind SOAP
Services Requestor
Services Provider
Service
Service description
Transport
Simple object access protocol (SOAP): Mechanism for connecting with applications and data.

Figure 10 High Level Web Services Architecture

1. First the user discovers all the possible Web Services from the UDDI registry and how to use them,
2. Then the user picks the best web service for their particular need.
3. From WSDL, the user now has details of how to invoke the desired service and can do so directly.

Note that, if particular modules occur together frequently, a user can select and group some services into a new, composite service that can be called and reused.

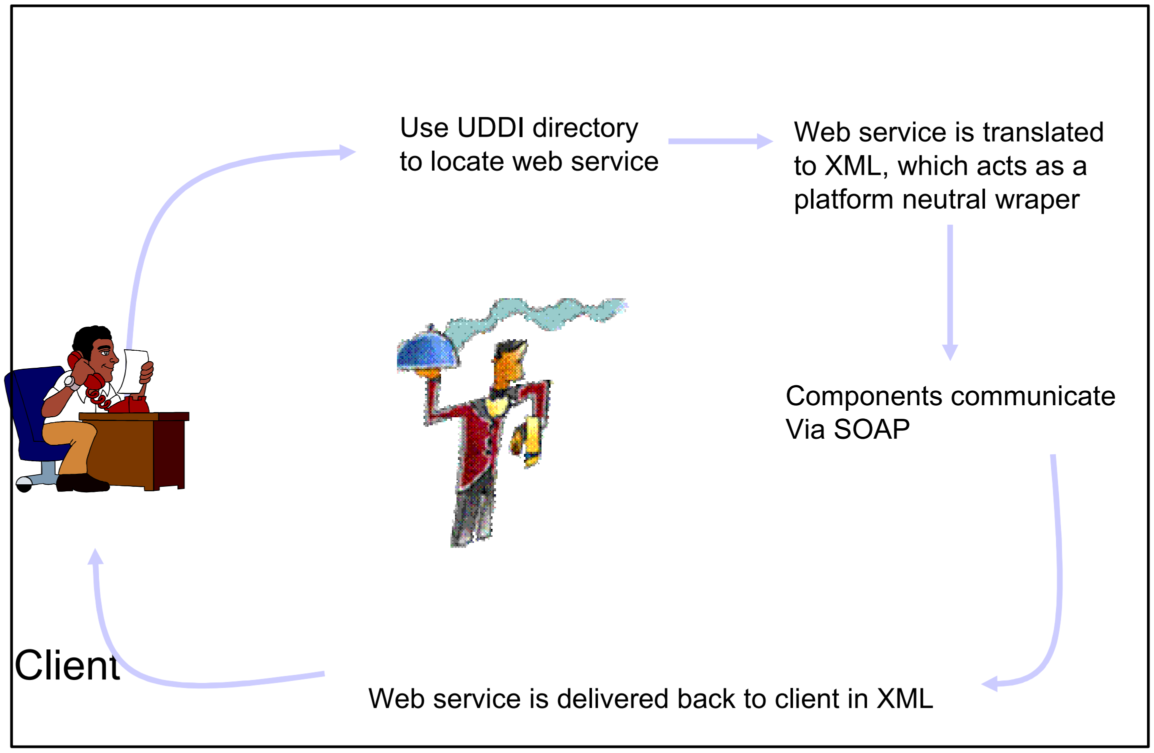
Use UDDI directory to locate web service
Web service is translated to XML, which acts as a platform neutral wraper
Components communicate Via SOAP
Web service is delivered back to client in XML
Client

Figure11. How Web Services Works

VIII. CONCLUSIONS AND LIMITATIONS

In this paper we presented the concept of Web Services from a stakeholder perspective, i.e, owner, architect, builder, and user. For each stakeholder, we present the salient issues and current developments such as dynamic business network, enterprise logic and modularity. We focused primarily on the promise of Web services and the particular value this approach holds for the four stakeholder perspectives.

We must also consider some of the challenges and drawbacks Web Services may entail for each of these stakeholders. In the following subsections, we discuss the limitations of Web Services.

OWNERS

From an owner's perspective, organizations can be modeled as dynamic business networks. Web Services add value by providing a framework suited for these dynamic networks. Note, however, that not all organizations are equally dynamic. A move to Web Services may not be cost-effective for organizations where tight system integration is more important than flexibility. In particular, the cost of adding flexibility to legacy systems may be difficult to justify.

Owners also face the question of who is responsible for the architecture, the components, and overall performance. Web Services alone cannot resolve these responsibility issues. Indeed, the increasing number of actors and organizations responsible for components of a functioning system under Web Services may actually exacerbate this problem.

DESIGNERS

From a designer's perspective, Web Services provide a framework for modular systems development. With Web services, however, designers must not only design the functionality to be provided by each module but must treat each module as a "product" that can ensure delivery of services to a wide range of clients across a wide range of circumstances. For example, Arsanjani et al [2003] identify such "nonfunctional" issues as performance, reliability, and level of service. This means that in addition to the required functionality, these “products” must also provide configuration toolkits that other designers can use to support their own unique requirements [Von Hippel and Katz 2003].

Finally, while the promise of Web Services rests, at least in part, on the wide adoption of key standards for interoperation, adopting such standards brings with it the risk that the market may switch to a different standard. This risk is especially significant in the current climate in which Web Services users are of necessity early adopters.

BUILDERS

From a builder’s perspective, Web Services provide a deployment framework for efficient development and interoperability of information systems using industry standards. For early adopters, however, there are significant performance and reliability issues to consider. For example Arsanjani et al [2003] note that the overhead involved in representing data using XML and text will result in "a data size explosion" and a significant processing overhead. In addition, early adopters have to deal with early versions of tools, limited infrastructure, and shifting standards. One may expect many of these issues to resolve themselves in the near future as additional tools and infrastructure are brought into play and performance and reliability problems are addressed. In the meantime however these are significant issues which builders should have consider.

USERS

From an end user’s perspective, Web Services promise an architecture for users to find relevant resources quickly and use them effectively. For the end user, however, facilities for the direct composition and manipulation of Web Services are not yet available. Instead, the end user is more likely to benefit from "trickle-down" flexibility that is enabled by designers and builders and implemented in the code. Perhaps more to the point, the infrastructure is not yet in place to provide a user with performance guarantees for specific Web Services. Finally, the search costs for end-users seeking to identify relevant Web Services are still significant, although such search costs will presumably continue to decline as the technology develops.

ACKNOWLEDGEMENT

Financial support from Boston University, School of Management, SMG Junior Faculty Development Grant is gratefully acknowledged.

Editor’s Note. This article, which was received on October 14, 2002, was fully peer reviewed. It was with the authors for approximately two months for two revisions and was published on April 23, 2003.

REFERENCES

Alexander, C. (1964) *Notes on the Synthesis of Form*, Boston: Harvard University Press.

Apache. (2002) "http://www.apache.org/," (current July 23, 2002).

Arsanjani, A.*, et al.* (2003) "Web Services: Promises and Compromises," *Queue* (1) 1, pp 48-58.

Balasubramanian, P.R., G. Wyner, and N. Joglekar (2002)"The Role of Coordination and Architecture in Supporting ASP Business Models," *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*

Baldwin, C.Y., and K.B. Clark (1997) "Managing in an Age of Modularity," *Harvard Business Review* (68), pp 73--109.

Baldwin, C.Y., and K.B. Clark (2000) *Design Rules: The Power of Modularity*, Cambridge, MA: The MIT Press, p. 471.

Birrell, A., and B. Nelson (1984) " Implementing Remote Procedure Calls," *ACM Transactions on Computer Systems* (2) 1.

Bovet, D., and J. Martha (2000) *Value Nets: Breaking the Supply Chain To Unlock Hidden Profits*, New York: Wiley p. 270.

BPMI. (2002) "http://www.bpmi.org/index.esp," (current July 24,2002).

Brandenburger, A., and B. Nalebuff (1996) *Co-opetition*, New York: Doubleday, p. 290.

Chandler, A.D. (1964) *Strategy and Structure: Chapters in the History of the American Industrial Enterprise*, Cambridge, MA: The MIT Press, p. 463.

Gawer, A., and M.A. Cusumano (2002) *Platform Leadership: How Intel, Microsoft, and Cisco Drive Industry Innovation*, Boston: Harvard Business School Press, p. 336.

Gaynor, M. (2002) *Network Services Investment Guide: Maximizing ROI in Uncertain Times*, New York: Wiley.

Gaynor, M., and S. Bradner (2001) "Using Real Options to Value Modularity in Standards," *Knowledge Technology & Policy* (14) 2.

Gaynor, M.*, et al.* (2001)"The Real Options Approach to Standards for Building Network-based Services," *IEEE conference on Standardization and Innovation*, Boulder CO IEEE.

Gnyawali, D.R., and R. Madhavan (2001) "Cooperative Networks and Competitive Dynamics: A Structural Embeddedness," *Academy of Management Review* (26) 3, pp 431-445.

Graham, S.*, et al.* (2002) *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI*, Indianapolis, IN: Sams, p. 581.

Hacki, R., and J. Lighton (2001) "The Future of the Networked Company," *The McKinsey Quarterly* (3), pp 26-39.

Hagel III, J., and J.S. Brown (2001) "Your Next IT Strategy," *Harvard Business Review,* (73:10) October, pp 105--113.

Joglekar, N., and P.R. Balasubramanian (2001)"Dynamics of Application Service Provision Business Models," *Proceedings of the International Conference of the Systems Dynamics Society*, Atlanta, GA.

Langlois, R.N. (1999) "Modularity in Technology and Organization," *Journal of Economic Behavior & Organization* (49)1, pp 19-37.

Linthicum, D. (2000) *Enterprise Application Integration*, Upper Saddle River, NJ: Addison-Wesley,p. 377.

Markus, M.L. (2000) "Paradigm Shifts - E-Business and Business/Systems Integration," *Communications of the Association for Information Systems* (4)10, pp 1--44.

McKeen, J.*, et al.* (2002) "IT Sourcing: Make, Buy or Market?," *Communications of the Association for Information Systems* (9) 8 September.

Morgan, T. (2002) *Business Rules and Information Systems*, Pearson Education, Inc., Boston, p. 348.

Nohria, N., and R.G. Eccles (eds.) (1992) *Networks and Organizations: Structure, Form and Action*. Boston: Harvard Business School Press.

Orfali, R., D. Harke, and J. Edwards (1996) *The Essential Client/Server Survival Guide*, New York: Wiley .

Parnas, D.L. (1972) "On the Criteria To Be Used in Decomposing Systems Into Modules," *Communications of the ACM* (15) 12, pp 1053-1058.

Schilling, M.A. (2000) "Toward a General Modular Systems Theory and its Application To Inter-Firm Product Modularity," *Academy of Management Review* (25), pp 312-334.

Simon, H.A. (1996) *The Sciences of the Artificial*, Cambridge, MA: The MIT Press,p. 231.

Sleeper, B. (2001) "Defining Web Services," San Francisco: The Stencil Group.

Tapscott, D., D. Ticoll, and A. Lowy (2000) *Digital Capital: Harnessing the Power of Business Webs*, Boston: Harvard Business School Press, p. 320.

TechMetrix (2002) "Which Technology for Tomorrow's EAI?," http://e-serv.ebizq.net/aps/techmetrix_2.html (current July 29, 2002).

Von Hippel, E., and R. Katz (2003) "Shifting Innovation to Users Via Toolkits," *Management Sciences* (48)7, pp 821-835.

WFMC. (2002) "http://www.wfmc.org/," (current July 24, 2002).

XML. (2002) "www.xml.org," (current July 23, 2002).

Zachman, J.A. (1987) "A Framework for Information Systems Architecture," *IBM Systems Journal* (26) 3, pp 276--292.

APPENDIX I. WEB SERVICE RESOURCES

GENERAL

www.webservices.org
www-106.ibm.com/developerworks/webservices
www.gotodotnet.com/team/XMLwebservices
www.w3.org/2001/01/WSWS
www.ws-i.org
www.ebizQ.net
www.eaiindustry.org

SOAP

www.w3.org/TR/SOAP
www.develop.com/soap/
www.soapware.org

XML

www.w3.org/XML/
www.xml.org
www.xmlrpc.com

WSDL

www.w3.org/TR/wsdl
xml.coverpages.org/wsdl.html

UDDI

www.ibm.com/services/uddi
www.uddi.org
www.uddicentral.com

COLLABORATION PROTOCOLS

www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf (WSFL)
http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/ (XLANG)
http://ifr.sap.com/wsci/ (WSCI)
http://www.bpmi.org/ (BPMI)

APPENDIX II. INFORMATION SYSTEM SOURCING

Regardless of the type of coupling between the entities in the network, information flows between systems in the network. An important problem encountered by owners is whether to own the information flows or outsource them [Lacity and Hirschheim,1993; Lacity and Willcocks, 1998; Hirschheim and Lacity, 2000]. A key finding is that sizeable investments are required to create the infrastructure necessary to integrate data and applications to sustain a viable network. Furthermore, controlling and setting the architectural standards is seen as more important than which applications a firm uses [Gawer and Cusumano 2002]

The owner's perspective results in the selection of a sourcing strategy – build, buy, rent, or share.To understand these alternatives, we must consider the evolution of software development practices. For almost all firms, their inventory of information systems was developed at different points in time and using resources both within and outside the boundary of the firm. IT service provisioning evolved through several delivery models. Traditionally, firms could rent, buy or build software applications to meet their business needs. Initially computers were an expensive, central resource requiring programming specialists. These initial computers were affordable only by the very largest corporations and were used primarily to support firm- specific, internally focused business needs through a 'dedicated services model'. Eventually, smaller organizations that could not afford the overhead of their own centralized computer resource could rent computer use from a third party. Renting applications was achieved through a 'shared services model' such as 'time sharing' or more recently through 'Application Service Providers' [Balasubramanian et al. 2002; Joglekar and Balasubramanian 2001]. Using the shared services model, individual firms rent the use of the application from a vendor who is responsible for developing and maintaining the hardware and software environment to support the user's needs.

The basis of this business model paradigm rests in sharing standard fixed costs (e.g., the data center, standard software licenses, and technical support/operations staff). The economics of sharing certain standard fixed costs across many customers provides advantage to both the vendor and their customers.

With more widespread use and continual reduction in the cost of computers, the emergence of generalized application software packages designed for specific industries or for specific functions was introduced. These software packages were designed to provide 80 to 90 percent of the user's needs, greatly reducing the amount of custom programming required. Firms now had the option of building custom applications or buying packaged applications. In addition, firms could rent application use through shared services. Applications have moved from centralized monolithic computing environments to distributed, multiprocessing and desktop computing. Packaged applications became more sophisticated and configurable to individual customer needs.

The computing software development paradigm however remained constant. Applications are designed for specific purposes in an a priori manner. Any flexibility must be 'in the design' and evoked through configurable switches or parameters. Typical application implementations require support from application programming specialists to customize the packaged standard code to meet the specific needs of an enterprise. The difficulty in this software development paradigm is that the applications are inherently difficult to change rapidly. The software is designed for a particular purpose. As a result, the typical enterprise will spend more money maintaining the software than they spend on the original implementation. Software maintenance and enhancement is typically the largest portion of an organization's IT budget.

Whether the software is rented, bought, or built, applications are developed and used by businesses for their particular customized purposes that are well defined in advance of their use.

Systems go through a careful design and acceptance process to assure compliance with an a priori understanding. Contractual agreements specify the form and function of the application. Given the development complexity, these applications are not designed to provide flexibility; rather the software is designed and run to meet specific fixed requirements.

This traditional approach to software development has been consistent with the business paradigm of the enterprise that the software was designed to support. Business focused on gaining competitive advantage through economies of scale, driving cost per unit down, and by increasing quality through the design and continual fine-tuning of very specific production processes. This view of the world results in tightly integrated systems. Tightly integrated systems sacrifice the ability to adapt rapidly to change (i.e., flexibility) for efficiency and/or effectiveness towards a particular purpose.

APPENDIX III. MODULAR DESIGN

Consider the evolution of a computer system without a modular design. Figure A-1 illustrates such a system: it performs 4 main tasks – storage, memory, I/O, and the CPU. Suppose that this computer is being redesigned and both the memory and the CPU are changed. Now, assume that this redesigned CPU worked well, and increased the value of the total system by +1, however the new memory design did not work as expected. It decreased the value of the total system by –2. When redesigning a computer whose functional pieces are interconnected, the new artifact provides a single choice; the new system performs as a whole either better, worse, or the same than its predecessor does. In this case, the value of the new system is less than the original system. They system is a failed memory experiment that drags down the total system value. The interconnected architecture of this computer does not allow the choice of only using the improved CPU, without the inferior new memory design.

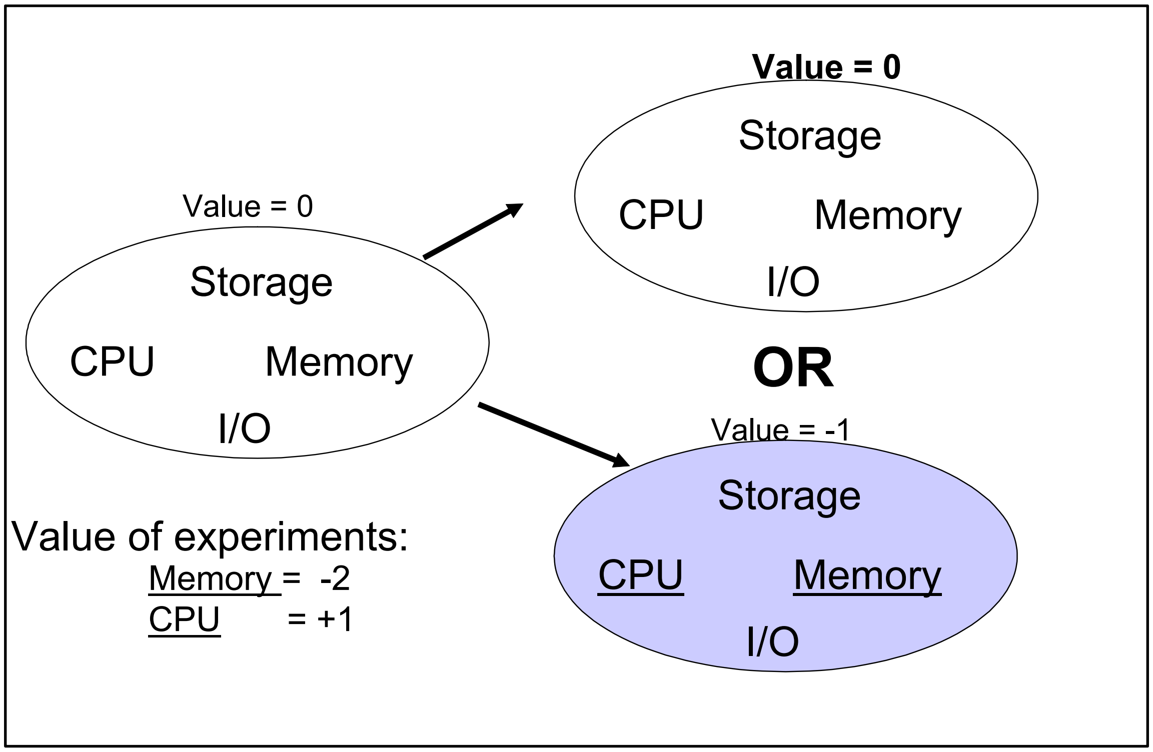

Figure A-1. Interconnected System

Figure A-2 illustrates a modular computer alternative. If the redesign is attempted with this modular architecture, many more choices are available for the new system compared to the interconnected system. The system with the most value only uses the new CPU, keeping the older, but better performing memory design. The value of this new system is +1, As with all options-like situations, increases in uncertainty increase the value of modularization. Modularity allows the system designer to pick and choose the components of the new system, thus maximizing the value. Uncertainty increases this value, because as it grows it increases the potential of a better choice emerging. Modular design increases value by providing a portfolio of options rather than a (less valuable) option on a single portfolio.

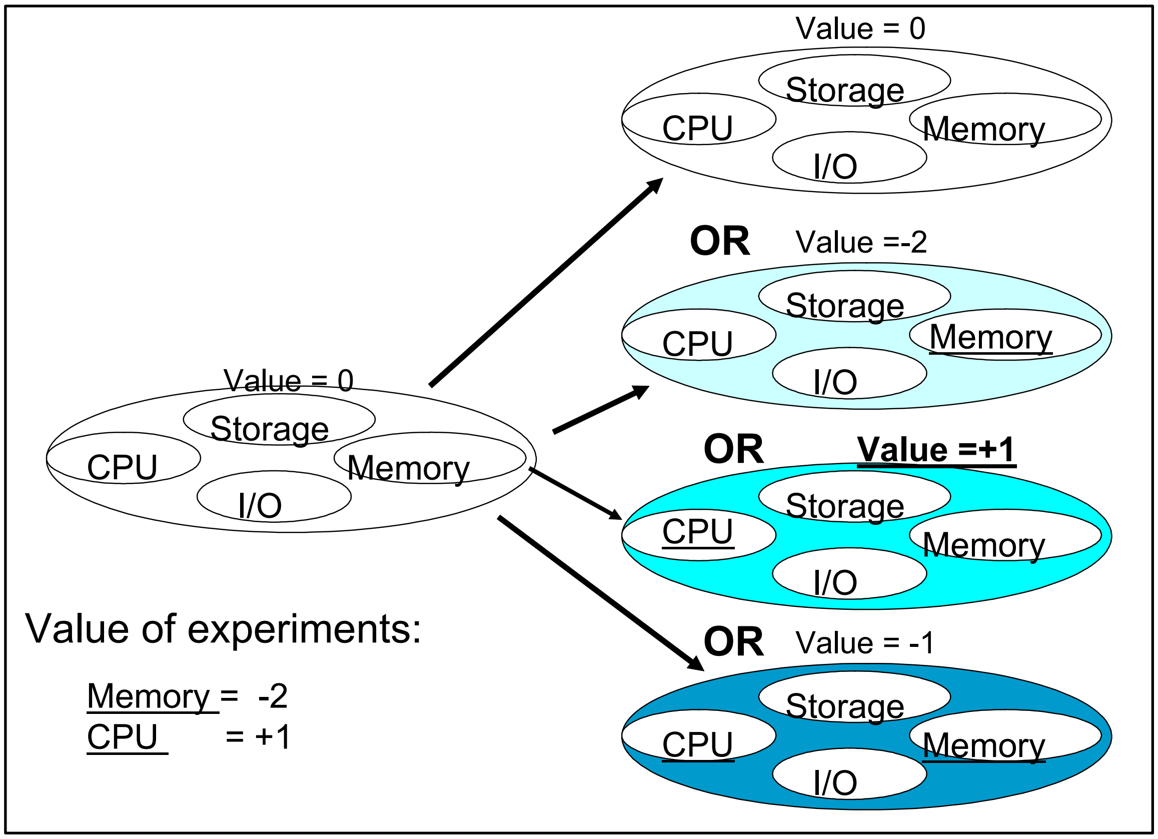

Figure A2. Value of Modular System

Modularization allows designers to experiment with the modules that have the most potential for altering the value of a system. Each experiment is one design of the system**.** Performing many experiments on the components most critical to overall system performance has the potential to improve the overall value of the entire system. The greater the technical uncertainty, the greater is the value for such focused experimentation. Because of the modular design, the designer can pick the best outcome from many trials.

For example, suppose the designers of a new computer system need to increase the rate a CPU module processes instructions (Figure A3) Three attempts are made to improve the CPU: the worst experiment lowers the value of the total system by –2, and the best new design increases the total system value by +2. By attempting several technically risky new technologies for a CPU, the designer can improve the odds of obtaining faster instruction execution. Modularity allows system designers to focus on components with the most potential to increase the value of the whole system.

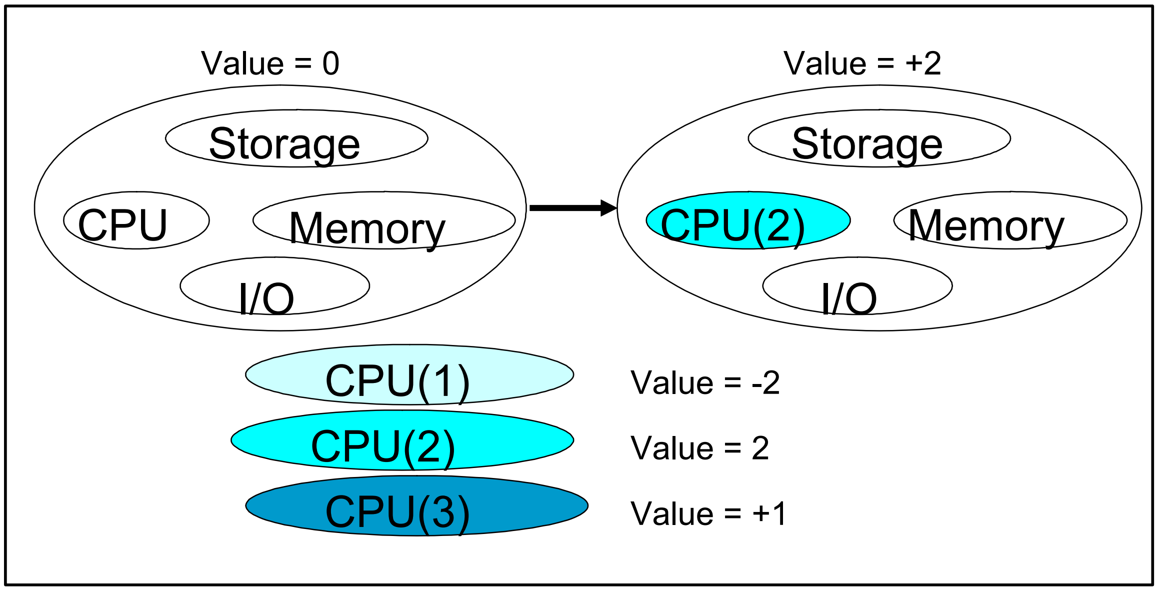

Figure A-3. Value of Experimentation

The value of picking the best module from the many choices that modularity allows is similar to the value gained from a distributed management structure, because the experimentation it allows enables users to select among many choices of network-based services. Interconnected systems make experimentation difficult in a similar way that central management hampers the ability to experiment. Modularity gives designers choices in the same way that distributed management gives users choices. Both approaches have the most value when uncertainty is high because of the increased value of experimentation enabled by the modular design, or the distributed management structure. This theory was extended to model modularity in IT standards [Gaynor and Bradner 2001] and general network services [Gaynor 2002, Gaynor et al. 2001].

LIST OF ACRONYMS AND THEIR MEANINGS

Acronym and Full Form	Meaning
.NET	both a business strategy from Microsoft and its collection of programming support for what are known as Web services, the ability to use the Web rather than your own computer for various services. Microsoft's goal is to provide individual and business users with a seamlessly interoperable and Web-enabled interface for applications and computing devices and to make computing activities increasingly Web browser-oriented. The .NET platform includes servers; building-block services, such as Web-based data storage; and device software. It also includes Passport, Microsoft's fill-in-the-form-only-once identity verification service.
Apache AXIS	An implementation of the SOAP ("Simple Object Access Protocol") submission to W3C.

API Application program interfaces	The specific method prescribed by a computer operating system or by an application program by which a programmer writing an application program can make requests of the operating system or another application.
Application "wrapping"	An application which wraps itself around ClockWatch to intercept and handle communication requests.
CORBA Common Object Request Broker Architecture	OMG's open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.
DCOM Distributed Component Object Model	a set of Microsoft concepts and program interfaces in which client program objects can request services from server program objects on other computers in a network. DCOM is based on the Component Object Model (COM), which provides a set of interfaces allowing clients and servers to communicate within the same computer (that is running Windows 95 or a later version
FTP File Transfer Protocol	A protocol used to request and transmit files over the Internet or other computer network
HTTP Hypertext Transfer Protocol	A protocol used to request and transmit files, especially webpages and webpage components, over the Internet or other computer network.
J2EE Java 2 Platform, Enterprise Edition	a Java platform designed for the mainframe-scale computing typical of large enterprises. Sun Microsystems (together with industry partners such as IBM) designed J2EE to simplify application development in a thin client tiered environment. J2EE simplifies application development and decreases the need for programming and programmer training by creating standardized, reusable modular components and by enabling the tier to handle many aspects of programming automatically.
Java	a programming language expressly designed for use in the distributed environment of the Internet. It was designed to have the "look and feel" of the C++ language, but it is simpler to use than C++ and enforces an object-oriented programming model. Java can be used to create complete applications that may run on a single computer or be distributed among servers and clients in a network. It can also be used to build a small application module or applet for use as part of a Web page. Applets make it possible for a Web page user to interact with the page.
NP Network Platform	A base of communication technologies on which other technologies, applications or processes are built.
OMG Object Management Group	is an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. Membership includes virtually every large company in the computer industry, and hundreds of smaller ones. Most of the companies that shape enterprise and Internet computing today are represented on the OMG Board of Directors.
RMI Remote Method Invocation	is a way that a programmer, using the Java programming language and development environment, can write object-oriented programming in which objects on different computers can interact in a distributed network. RMI is the Java version of what is generally known as a remote procedure call (RPC), but with the ability to pass one or more objects along with the request. The object

	can include information that will change the service that is performed in the remote computer.
RPC Remote Procedure Calls	is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details. (A *procedure call* is also sometimes known as a *function call* or a *subroutine call*.)
SMTP Simple Mail Transfer Protocol	a protocol for requesting and transmitting mail documents
SOAP Simple Object Access Protocol	SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework
UDDI Universal Description, Discovery and Integration	a "meta service" for locating web services by enabling robust queries against rich metadata
WSDL Web Services Description Language	WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate
WSFL	The Web Services Flow Language is an XML language for describing Web Services compositions.
XML Extensible Markup Language	A metalanguage written in SGML that allows one to design a markup language, used to allow for the easy interchange of documents on the World Wide Web.

ABOUT THE AUTHORS

Bala Iyer is assistant professor of Information Systems in the department of information systems, Boston University. Professor Iyer received his Ph.D. from New York University in MIS with a minor in computer science. His research interests include architecture design, knowledge management, query-driven simulation and its applications in manufacturing, and hypermedia design and development. His papers appear in the *Communications of the ACM, Decision Support Systems, Annals of Operations Research, Journal of Strategic Information Systems, Communications of AIS* and in several proceeding of the Hawaii International Conference of Systems Sciences. He serves on the editorial board of the Journal of Database Management.

Jim Freedman Jim has over 28 years experience as an information technology consultant and educator. He worked with clients in the financial services, manufacturing, distribution, retail, transportation and software industries. His experience spans a wide range of computing platforms since he began as a programmer supporting applications running on mainframe computers. He is currently on the faculty at Babson College, teaching graduate level courses in e-

business and information systems management and is a doctoral candidate at Boston University. His research interests are in the adoption of emerging and disruptive technologies.

Mark Gaynor holds a Ph.D. in Computer Science from Harvard University and is Assistant Professor in the Graduate School of Management at Boston University. His research interests include ATM, TCP/IP over high speed ATM networks, packet classification for Quality of Service, standardization in the IT area, designing network based-services, and wireless Internet services. He is on the research board of Telecom City (a regional technology development project) and is a Co-Principal Investigator on an NSF Grant studying virtual markets on a wireless grid. He is the technical director and network architect at 10Blade. His book, *Network Services Investment Guide: Maximizing ROI in Uncertain Markets*, is being published by Wiley this year.

George Wyner is Assistant Professor of Information Systems at the Boston University School of Management. Professor Wyner received a Ph.D. in Management from the Sloan School of Management at MIT. His research centers on the modeling, classification, and analysis of organizational processes to achieve systematic technology-enabled organizational innovation. His papers appear in *Management Science, Information Systems*, and in the proceedings of the International Conference on Information Systems.

Communications of the Association for Information Systems

ISSN: 1529-3181

EDITOR-IN-CHIEF
Paul Gray
Claremont Graduate University

AIS SENIOR EDITORIAL BOARD

Cynthia Beath Vice President Publications University of Texas at Austin	Paul Gray Editor, CAIS Claremont Graduate University	Sirkka Jarvenpaa Editor, JAIS University of Texas at Austin
Edward A. Stohr Editor-at-Large Stevens Inst. of Technology	Blake Ives Editor, Electronic Publications University of Houston	Reagan Ramsower Editor, ISWorld Net Baylor University

CAIS ADVISORY BOARD

Gordon Davis University of Minnesota	Ken Kraemer Univ. of California at Irvine	Richard Mason Southern Methodist University
Jay Nunamaker University of Arizona	Henk Sol Delft University	Ralph Sprague University of Hawaii

CAIS SENIOR EDITORS

Steve Alter U. of San Francisco	Chris Holland Manchester Business School, UK	Jaak Jurison Fordham University	Jerry Luftman Stevens Institute of Technology

CAIS EDITORIAL BOARD

Tung Bui University of Hawaii	H. Michael Chung California State Univ.	Candace Deans University of Richmond	Donna Dufner U.of Nebraska -Omaha
Omar El Sawy University of Southern California	Ali Farhoomand The University of Hong Kong, China	Jane Fedorowicz Bentley College	Brent Gallupe Queens University, Canada
Robert L. Glass Computing Trends	Sy Goodman Georgia Institute of Technology	Joze Gricar University of Maribor Slovenia	Ruth Guthrie California State Univ.
Juhani Iivari University of Oulu Finland	Munir Mandviwalla Temple University	M.Lynne Markus Bentley College	Don McCubbrey University of Denver
Michael Myers University of Auckland, New Zealand	Seev Neumann Tel Aviv University, Israel	Hung Kook Park Sangmyung University, Korea	Dan Power University of Northern Iowa
Nicolau Reinhardt University of Sao Paulo, Brazil	Maung Sein Agder University College, Norway	Carol Saunders University of Central Florida	Peter Seddon University of Melbourne Australia
Doug Vogel City University of Hong Kong, China	Hugh Watson University of Georgia	Rolf Wigand University of Arkansas	Peter Wolcott University of Nebraska-Omaha

ADMINISTRATIVE PERSONNEL

Eph McLean AIS, Executive Director Georgia State University	Samantha Spears Subscriptions Manager Georgia State University	Reagan Ramsower Publisher, CAIS Baylor University