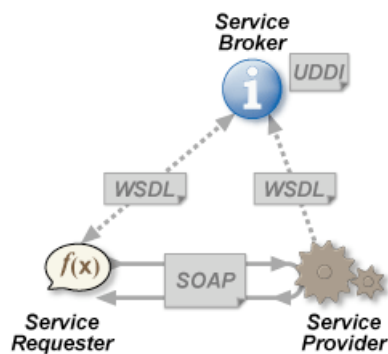


New Perspective On Web Services

Web services are thought of as both foundational to the next technology architecture and as an extension of the IT infrastructure that is currently in place. When will IT departments and infrastructure vendors who view web services as a means to extend enterprises as we know them today see web services instead as the organizational and technical enablers that they are? When will IT and infrastructure vendors admit that the only way to effectively leverage web services to create a business entity that can operate on a global basis is to embrace a new point of view?

Web services represent a paradigm shift unlike others we have seen in the technology market in the past 20+ years.

As we consider paradigm shifts from mainframe-based, client/server, distributed object and component-based computing, we observe that the boundary between business goals/objectives/strategies and technical provisioning of the same have not been impacted by them: these paradigm shifts were technology-focused only and left the business model hard-wired into software applications. Best of breed applications, integrated as best we could with enterprise application integration technologies proved only marginally successful in enabling business entities to reclaim ownership of business models and ability to adeptly change them. Infrastructure drove applications and the ways they provisioned functionality in a way that was unaccommodating of business change.



Web services enable greater visibility to and traceability of business functionality from the business model to its technical implementation that, in turn, may be viewed agnostically from a technology point of view. Consequently, application architecture and infrastructure can be positioned to drive business models even less. We have learned over the past years that separation of “business” rules from application code increases a company’s ability to change in response to the needs of special partnerships or industry transitions. Web service fabrics, emerging technologies that enable management of web services, refer to business and operational rule sets as *policies* and use them as constraints on the ways that web services are combined to provision business and operational functionality. The ways in which policies are used to factor business and operational intelligence out of technology components enable a business to flexibly modify its business model and supporting technology without the need to modify code in every case. Policies enable change to be localized and systematically managed without compromising agility. A business that positions itself to move agilely yet systematically positions itself to rapidly evolve and/or transform itself through the development of new business capabilities.

Leveraging web services to a business entity’s full advantage requires the business entity to view web services as a kind of inflection point. An inflection point represents a time of turbulence and instability. The web services market certainly is unstable at present.

There are – by our count of vendors that we would consider *significant players* in the market – more than 35 vendors who offer software and hardware products that serve as components in a web services platform. While only a small number of products are required to put together a web service platform, the fact remains that the buyer must research, purchase, and deploy products from multiple vendors. In some cases it is necessary to compensate for functionality that does not yet exist.

To further underscore the point, we note that web service standards and standards proposals are in a state of flux. Proposals that have become bona fide standards have not been quickly adopted (e.g. UDDI is viewed as a critical web services standard, yet UDDIv3 has not been quickly adopted, and companies are uncertain of its value in light of the fact that they cannot look at their own software assets and point to a large enough number of services significant to warrant management using some form of design or runtime registry).

So why should CIOs adopt a web service-based technology strategy now? Or – more appropriately – what will CIOs be unable to do if a web service-based technology strategy is not adopted?

Chief among the goals to be realized through adoption of a web services strategy is the ability to position the enterprise to rapidly create new capabilities – whether within or outside of what we think of today as *the enterprise*. The near universal support for use of XML as the common syntax for message exchange and the use of http as a common transport protocol provides enterprises with the abilities to commoditize application technology integration within an enterprise and collaborate more easily with internal as well as external business partners. These abilities enable business institutions to both tactically optimize their margins *and* strategically evolve their business models. Common syntax and protocol are required to: rapidly specialize business interactions without compromising the ability to systematically manage interactions in general; clearly communicate and coordinate business interactions at human *and* technology levels so that business goals can be traced to technology as necessary, and the value propositions of business relationships can be effectively measured and monitored; and leverage current capabilities to create new capabilities that span institutional boundaries. These, in turn, empower business institutions with the ability to introspect and reshape themselves as a function of their desire to innovatively seek out opportunities to create new revenue streams and more rapidly position themselves to be seen as and participate in the business flows of a growing and global market.

But simple-minded adoption of web services is no silver bullet – technology never is. As business institutions strive to realize the goal of more rapid creation of new capabilities using web services, the necessity to question the shape of a business model, the ways that business should be conducted, and the ways that web services should be leveraged should not be surprising. For example:

- ❑ Automation of business processes within an enterprise has been strongly influenced by the business application infrastructure that the enterprise selects and deploys, and is therefore limiting with respect to multi-party business interactions that span institution boundaries. Is this by necessity? Or has this become the norm because it is a path of least resistance?
- ❑ Consider the way that enterprise applications are made to interoperate. Do enterprise application integration platforms we commonly find in enterprises today represent a kind of *lowest common denominator* collection of capabilities that we would expect to find in *all* business software platforms? If so, and the complexity that businesses see in their platforms today is *the norm*, how can businesses create new capabilities *rapidly*? Are businesses condemned to throw bodies that cost less at developing new software functionality and hope that *agile software development* one day pans out?
- ❑ Consider an enterprise, as it exists today, as a centralized locus or hub of control. Is that model realistic given what appears to be a growing requirement to rapidly construct and tear down virtual enterprises as communities of practice that interact to achieve some common business goals and then disband?

Peter Drucker observes that "the problem in times of turbulence is not the turbulence; it is acting with yesterday's logic." The goal to more rapidly create new business capabilities in global contexts is a formidable goal. Simply applying/extending "yesterday's [enterprise integration] logic" to adoption of web services without revisiting the foundation of that logic is ill advised. We believe there are two points of view one could take toward web service adoption that will be helpful in determining a go-forward position vis-à-vis adoption of web services as a business strategy component.

The first, which we call *in-out*, is the commonly held point of view (pov) arguing that web services should be implemented by extending what is currently inside today's enterprise outward to support multi-party business interactions that span enterprise boundaries. The in-out pov is one that accepts web services as a technology evolution from the *inside* of enterprises (as we know them today) *out*, permitting co-existence of enterprise assets like J2EE EJBs and Microsoft ActiveX components as manageable functional elements. It encourages adoption of web services as a means to encapsulate complexity rather than eliminate it. It does not challenge traditional thinking about enterprise boundaries – and it encourages the agile implementation of new business models in global economies. It represents the belief that current business practices and operational models will extend beyond the enterprise when necessary, and an expectation that IT will be able to dress up its infrastructure in some way that will enable the business to meet global demands that the business now considers to be strategic. It is a commonly held bottom-up pov.

The second pov, which we call *out-in*, argues that existing application and integration infrastructure will not scale outward in a cost effective way. The *out-in* pov requires business to be viewed from a business capability or *service only* pov to eliminate many of the challenges that heterogeneous IT infrastructure causes, and to push beneath a standard interface layer the technologies that cannot or should not be standardized. One potential result of assuming such a viewpoint is the ability to eliminate certain infrastructure components. Homogeneity and elimination of infrastructure components simplifies the platform according to this pov. Another potential result is that how IT provisions services to employee or community-of-practice desktops/PDAs can be rethought (it is no longer necessary to deliver a vendor's "application" to either of these). Another is that business service development could be accomplished as a combination of raw material web service development and composite service-based application assembly – but the future applications more and more will be composite and web services based. It is very much a top-down pov that has been broadened to scale beyond traditional institutional boundaries.

The out-in pov seems considerably risky relative to the devil-you-know in-out pov UNLESS you see web services as an inflection point where business model, organization, and technology converge. Web services can be considered a means to transition a business entity in its entirety to be service-based (out-in) rather than simply as a means to optimize costs at an IT bottom-line (in-out). One can argue that either pov can be adopted to transition to being web service-based. We would concede this point. However, we would do so only as we observe that the cost of taking the in-out pov represents business opportunities that would be lost en route, and that losing such opportunities could prove fatal.

This paper introduces the out-in pov by examining limitations of the more common in-out pov in the context of global enterprise enablement. It does this first by considering technical limitations of enterprise infrastructure that is being extended by a web services layer. Next, it examines out-in implications on architecture, and on organization. Finally, it examines the benefits and detriments of the point of view.

The out-in pov is considered new in this paper because it takes a technology agnostic view to whatever is below a standards-based web services interface boundary – without denying the fact that legacy systems will exist for the foreseeable future and must be included as a component in any architecture strategy. One can argue that Web Service proponents take this technology agnostic position already. However, we believe few if any vendors or enterprises have given more than intellectual assent to the value of taking such a view.

1. TECHNICAL DISTINCTIONS BETWEEN OUT-IN AND IN-OUT

There are two points of view an architect could take when architecting a web service-oriented platform: (a) one in which the platform manages the life cycle of a web service – the only functional component that is exposed; and (b) one that extends an existing architecture to manage web services in addition to other types of functional components.

(a) represents a "purist" point of view (pov) on a web service-oriented architecture, and we label it *out-in*. Web services represent the *only* functional component that is created and managed in this architecture.

Architecture objectives justifying this pov include avoidance of architecture erosion and drift.

Perry and Wolf discuss architecture erosion and drift in *Foundations for the Study of Software Architecture* [2], both of which are common architecture problems. Erosion is due to violation of architecture over time. Drift is due to insensitivity to the architecture as it was originally designed. Both result in brittleness and unnecessary increased complexity of systems if not appropriately managed. Both can be minimized with respect to technology *if* a web service-oriented architecture is kept simple by focusing it on web services *only*.

The *out-in* pov represents the *outside-in* and *top-down* view of web service-based applications built on a web service-oriented platform. It acknowledges that there are numerous technologies with which web services can be constructed and deployed (e.g. a variety of technology stacks can be used), but it avoids making these technologies and their complexities *components* that must be managed directly by the web services-oriented platform. The ultimate goal of taking this pov is to ensure technology agnosticism above and including a web services standards-based interface boundary.

The out-in pov is based upon conjecture that what the industry considers to be good enterprise architectures today are not suitable to meet demands of next generation enterprises that interact using web services.

(b) represents the pov called *in-out* because it *extends* technologies already in use within enterprises today to manage the life cycle of web services. “Extends” implies that the architecture does not limit its functional components to web services only.

The *in-out* pov represents a *bottom-up* view of web services. Web services are constructed using EAI, J2EE, database and other technology stacks, and these technology stacks must participate in web service life cycle management just as they participate in the management of other components developed using them (e.g. A J2EE technology stack must manage web services just as it must manage EJBs).

The in-out pov is based upon conjecture that what the industry considers to be good enterprise architectures today can be extended to manage the web service life cycle, and that these architectures will prove suitable to meet demands of next generation enterprises that interact using web services.

To better understand the differences between the two pavs, we consider three areas where there is architectural difference between them: transaction management, exception handling, activity granularity, and orchestration.

1.1 Transaction Management

The short-lived *unit of work* that we commonly see inside the enterprise today is referred to as an *XA-compliant transaction* [3]. It is defined as a set of software activities that are executed as a unit to cause a change of state in the transaction execution environment. Some of these activities have persistent side effects and have ACID properties:

- ❑ (Atomtic) Activities with side effects either succeed or fail together. When failure occurs, effects of all activities should be undone, and the state of the execution environment should be rolled back to its previous state;
- ❑ (Consistent) Unit of work activities transition the business from one consistent state to another;
- ❑ (Isolated) Resource changes effected by unit of work activities are not shared until the unit of work completes; and
- ❑ (Durable) Once a unit of work completes, its effects are guaranteed despite any business infrastructure failures.

It is desirable that a long-lived unit of work also should have ACID properties – though it is not possible to make it do so without relaxing the definitions of these properties:

- ❑ (Atomic) It may be impossible or undesirable to roll back all the side effects of a long-lived unit of work. A long-lived unit of work defines the essential activities for which compensation plans must be defined/executed if failure conditions occur while potentially still leaving some side effects in existence.
- ❑ (Consistent) Unit of work activities transition the business from one consistent state to another; roll-back restores a consistent state, though not necessarily the state the execution environment was in at the time the long-lived unit of work started.
- ❑ (Isolated) Resource changes effected by unit of work activities are not shared until the unit of work designates this information as sharable and/or successfully completes.
- ❑ (Durable) Once a unit of work completes, its effects are guaranteed despite any business infrastructure failures.

The duration of a unit of work greatly impacts how each of the ACID properties like atomicity and isolation can be implemented. If the duration of a unit of work is short-lived, and assuming resources in the unit of work are XA-compliant, then operating system concepts like critical resource locking can be applied to ensure concurrently executing units of work do not share transient information. If the duration is long, an alternate strategy to keep intermediate results private is necessary since, as with operating systems, it is costly to lock critical resources for extended periods. If all resources managed in a unit of work implement a standard unit-of-work interface like X/Open's XA interface, then the atomic property can be implemented as 2 Phase Commit. If resources do not conform to such a standard interface, then an alternative way to commit or rollback unit of work effects must be implemented.

ACID properties are often considered together with granularity of unit of work activities and the order in which these activities can be performed. Granularity of work is important because it can influence things like unit of work duration and resource locking. Order of execution is influenced by business rules and constraints on information manipulated in the unit of work. In the past, order of execution was *hardwired* in 3/4GL code; but it can be treated more as data today with the advent of work- and process-flow technologies.

Because the concepts of *ACID*, *granularity of work*, and *order of execution* are interlaced, we refer to all of them using the acronym *UoW*. And we use *UoW* as a foundation for evaluating the in-out and out-in povs on *work* as it can be accomplished using Web Services.

The following definition is needed for subsequent discussion:

A business interaction is a set of coordinated business activities performed to realize a specific business objective. A business interaction is a UoW. A business interaction activity may be another business interaction.

The concept of work as a collection of activities having ACID properties is common to both in-out and out-in. However, there are tremendous differences in the ways that work is defined, and in the environments in which work is conducted. Differences are summarized in figure 1:

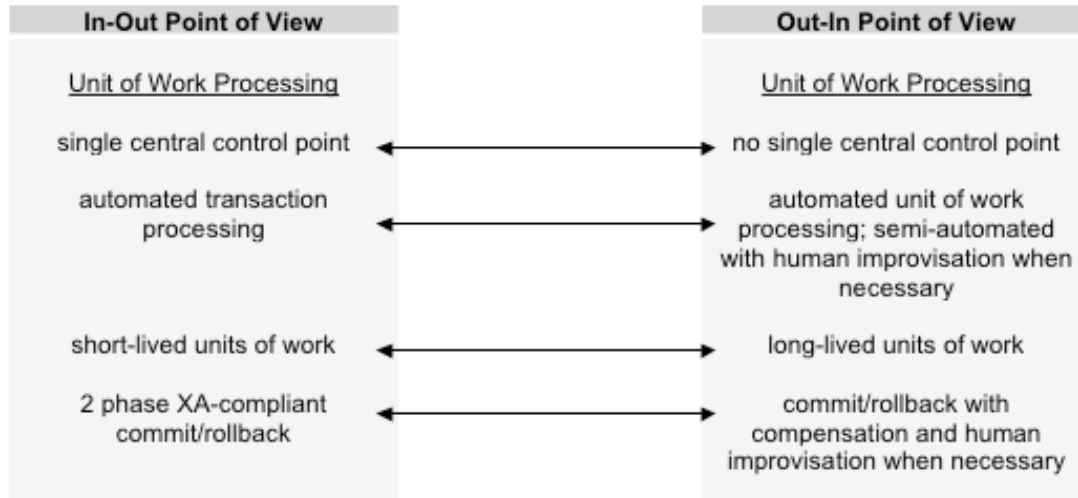


Figure 1

Enterprise applications commonly are constructed to interact with relational databases transactionally. Paradigm shifts in computing like Client/Server Computing and Object Oriented Programming have transitioned us from building monolithic applications to developing distributed object- and component-based enterprise application systems. But these paradigm shifts have not affected the definition or implementation of an application UoW as we have come to know it in any fundamental way. A UoW, even in an application server-enabled enterprise context, continues to be implemented as a short-lived, message-oriented, XA-compliant (possibly nested) transaction. It is this characterization of UoW that equates to the in-out UoW.

An application server uses a TP Monitor to manage the life cycle of a transactional application UoW that may be distributed across traditional application boundaries, and over a heterogeneous set of databases, queues, or message transports. The TP Monitor's purpose is to ensure that the UoW processes completely or, if an error occurs, appropriate actions are taken as follows:

(Atomic) Using techniques like journaling (before/after snapshots), the TP Monitor manages (commit/rollback) state in XA-compliant persistent stores.

(Consistent) The TP Monitor ensures that all transactional UoW activities represent a correct transformation of the transactional system that it manages. It ensures that the UoW as a whole satisfies all constraints on the transactional system, else it does not permit the UoW to successfully complete.

(Isolated) The TP Monitor coordinates work across Transaction Contexts and the Transaction Objects (or Resources) they manage to ensure that transaction-related information is not visible outside of Transaction Context boundaries until transactions are successfully completed. The TP Monitor uses techniques like short-term critical resource locking to implement isolation.

(Durable) The TP Monitor uses logging to ensure that changes to a system made in a committed transaction are not lost even if the servers on which it and the transactional system are running crash afterwards.

An out-in UoW is almost the antithesis of an in-out UoW. It is a business interaction that is long-lived, document-oriented, nested, and not XA-compliant. While an in-out UoW is designed to be automated, an out-in UoW involves both human and system participants. An in-out UoW is managed within enterprise boundaries or in special (IP tunneling¹) cases that extend the enterprise boundaries, but an out-in UoW

¹ A technology that enables one IP network to send its data via another IP network's connections. Tunneling works by encapsulating a

must be managed across enterprise boundaries as the normal rule.

ACID properties in an out-in UoW must be supported as follows:

(Atomic) Failure must be dealt with in a compensational manner since there is no analog to XA compliance in out-in. Compensation refers to a set of activities that either reverses the effects of essential interaction activities performed up to a point of failure and causes the interaction to halt, or corrects the problem that triggered the exception and causes the interaction to continue. The long-lived nature of out-in interactions underscores the importance of capturing interaction state and goals so that if necessary a human being can understand what has transpired up to and including interaction failure, and to determine how best to fix the problem causing the exception condition.

(Consistent) Each interaction must be expressed in the form of goals the interaction is to achieve, together with participant contracts that enable participant coordination. Goals must be structured as business rules and constraints so that consistency of state may be maintained, and so that participants, whether process or human, can use them to know how to participate in the interaction.

(Isolated) Interaction content and state must be managed so that it is not inappropriately shared before the outmost interaction completes². There is potentially a need to declare within the definition of an interaction when certain information can be shared.

NOTE: While it will be possible to use existing enterprise applications to provision Web Services, it will be necessary to modify them or to develop middleware layers to encapsulate them if intermediate interaction state is to be kept private until an interaction completes. These applications do not organize/partition the information that they manipulate by services. Consequently, information may be shared to other application components (e.g. using stored procedures and triggers) in a way that violates isolation principles. Completely addressing this would require a full rewrite of applications used to provision services (generally considered to be impractical if possible at all).

(Durable) Durability of an out-in UoW means that changes made to the execution environment by an interaction that successfully completes must be guaranteed despite any business infrastructure failures. Aside from business-related changes, this includes state changes of the interaction UoW itself.

Durability also means that completed interaction results must be reliably communicated to all interaction participants who wish to have them, for whatever reason. If these results cannot be reliably communicated, then compensatory activities must be triggered to roll the interaction back. NOTE: Infrastructure that oversees the interaction cannot be obligated to guarantee that participants successfully process interaction results since results might not be processed on a timely enough basis (e.g. they could be processed on a batch basis).

The differences in short- and long-lived UoWs illustrate that the infrastructure used to manage in-out UoWs is not designed to meet the requirements of out-in UoWs. In-out infrastructure is designed to manage a UoW and invoke exception management after a fault occurs to automatically restore state to what it was before the UoW started. Out-in infrastructure must be designed to manage exception management, implemented in code or through human mediation, as the UoW is in process.

network protocol within packets carried by the second network.

² Services invoked within an interaction may actually be interactions themselves (analogous to a nested OLTP transaction).

1.1.1 EXCEPTION HANDLING

When the phrase “exception handling” is mentioned in software technology-related conversation, it usually references managing faults within code. Software developers *raise* an exception in their code when some fault occurs – and an exception handler (software) *catches* it and attempts to roll back the effects of work performed from the start of a unit of work until the point the exception was raised. In the case of an XA Transaction, rollback is performed with the help of a TP Monitor. Without a TP Monitor, software developers must develop their own rollback mechanisms.

The exception usually includes some kind of exception identifier and brief text that attempts (if you’re lucky) to explain what went wrong and why to a human at some console or using some dialog box. It may include additional information that software might use within the context of work rollback, but most of the time information packaged in an exception is logged to some system file as a record that the software system noted an exception and took some corresponding action (even if the only action that it took was to write the exception to a log file).

The distinction between out-in and in-out with respect to exception handling is that exceptions cannot by default simply be rolled back automatically in an out-in pov as they are in an in-out pov. Long-lived business interactions may occur over months or even years. Rolling such interactions back might be impossible – or, were rollback possible, might not be desirable. Further, capturing in detail the steps necessary to compensate for a fault in a long-running interaction may require the help of a human being – not just to read after-the-fact an explanation from a software developer about a problem that an application system encountered and reported, but to participate in problem resolution at the business level all the way to a technology level as necessary. According to the in-out pov, it is acceptable to raise an exception, roll back work, and tell someone (human or log) that work it attempted to do could not be done.

1.2 Activity Granularity

Enterprise applications manage their own state and usually make state visible through an application-specific front-end (sometimes graphical, sometimes programmatic) user interface. When such applications are integrated, integrations may be implemented as follows:

- ❑ Messages are fine-grained packets of information that contain only the data and subset of application state each communicating application requires to map the message into its own context. These information packets are exchanged between communicating applications to realize real- or near real-time application integration.
- ❑ Message structure is formed by generalizing data structures from each application being integrated so that they can be semantically mapped to application specific data structures as required.
- ❑ The order in which messages are sent is determined by the application that is assigned the role of master in the integration.
- ❑ Messages are delivered between applications through a messaging framework that is implemented as a direct point-to-point API integration, a queuing framework, or as object messaging or EAI frameworks. Order of delivery is codified in the form of a procedural script that is managed by the messaging framework.
- ❑ Exception messages that occur due to technical failures in such communications include information from all communicating applications together with infrastructure-related failure data so that application-knowledgeable technologists can debug and problem solve.
- ❑ Exception messages that occur due to business failures are communicated to a business

analyst/specialist through some form of application inbox or workbench (the user interface) so that business issues may be addressed.

Application messages sometimes are called documents because they are rendered using XML. However, the kind of information packet exchanged in application-to-application integrations is not the kind that is exchanged in multi-party business interactions, nor could it be for the following reasons:

- ❑ All parties involved in an interaction do not use the same enterprise applications. Exchanging messages between disparate applications to support interactions with parties $\gg 2$ is an n^2 problem that quickly becomes unmanageable and unaffordable to implement at an out-in scale. Applying the logic that “there are only so many (read as ‘some reasonably small and finite number of’) applications that must be integrated” to this n^2 problem is flawed reasoning once application versions and upgrade strategies are considered.
- ❑ Full documents (e.g. purchase orders, mortgage and legal documents, faxes and financial trades) are exchanged in multi-party business interactions, as compared to application and object messages exchanged in enterprise application integrations. In many cases, business analysts must legally provide data to an interaction using specific documents and forms if the business enterprises they represent are to be held accountable for their participation. In other cases, communities of practice simply choose to interact with standardized forms to minimize training and limit errors.
- ❑ Application messages do not contain full interaction state (since applications maintain this state), whereas interaction documents do. Improvisational contributions to the interaction, as well as exception management would be, at best, difficult without context and state information.

The coarse-grained information that includes interaction context and state exchanged in out-in interactions is referred to as an interaction document because there is a direct correlation between this type of information set and the type managed by people in the process of conducting business. The gap between a message and a document may be referred to as an impedance mismatch³. A relatively significant investment would be required by enterprise application and application integration vendors to address this mismatch since doing so equates to grafting a business document model onto an application message-based document model, and localizing and isolating application data and process models.

The in-out pov of Web Services does not generalize well beyond enterprise boundaries because of the cost implied to retrofit service- and document-oriented thinking onto applications not architected with services in mind.

1.3 Orchestration

Business entities obviously make different choices concerning the technology and application infrastructure that they use to provision the functionality required to conduct business. While many business functions performed by these businesses are the same, implementations of these functions differ at process/task and data-entity levels because of the application and technology choices that have been made.

To coordinate a multi-party interaction from an in-out pov, it is necessary to build a collection of enterprise application integrations for all partners that must communicate. This means it is necessary to expose the details of processes embedded in participants’ applications, together with data structures that must be reconciled in order to communicate through the integration. An integration for each pair of communicating

³ Impedance mismatch is a problem in electrical engineering that occurs when two transmission lines or circuits with different impedances are connected. This can cause various losses and noise. In programming terminology, it refers to the attempt to connect two systems that have very different conceptual bases, a common example being use of a SQL database from an object oriented program.

applications must be built and maintained as participants make changes to their infrastructure, and as participants come and go.

The in-out pov forces participants to build point-to-point application integrations or to adopt canonical process and data models in order to participate in multi-party interactions. In practice, getting agreement at a canonical data structure level is not as formidable a task as getting agreement at a canonical process level (unless agreement comes at a very high level) since the latter implies the former and requires the development of additional middleware (e.g. adapters) to reconcile process-specific differences.

The out-in pov stipulates that there must be agreement upon interaction outcome, but agreement to adopt a common process to realize this outcome is not required. Outcome and the ways in which the outcome may be realized are separated based upon the beliefs that processes are enterprise implementation specific and, if propagated beyond enterprise boundaries, implementation details will severely limit the manageability and scalability of interactions as participants come and go.

Agreement on interaction outcome is expressed in the form of an interaction document that is used to coordinate participants as a function of constraints on the interaction. The interaction document captures interaction context and state, constraints and business rules that define interaction goals and are used to measure progress toward realizing those goals, contracts that participants must fulfill to realize interaction goals, and business content.

Constraints can be used to represent entire families of procedurally expressed business and data collection processes, where the word *family* connotes procedurally scripted processes that all produce the same business results regardless of enterprise applications and technology specifics used to produce them. Constraints, though nondeterministic, can be bounded so that they can be used to realize independence from business and data collection process specifics prescribed by provisioning enterprise applications. This circumvents the need for all business interaction participants to agree to a specific process at the level of fine-grained activities. And constraint satisfaction can be used to support various models for participating in an interaction. For example: the consequences of the constraint satisfaction could be published in the form of events to subscribing participants (kinds of event listeners), thus triggering participants to take action.

The in-out pov of Web Services is more application or process-centric than outcome-centric, limiting its scalability to the enterprise. The out-in pov requires agreement upon interaction outcome, but agreement to adopt a common process to realize this outcome is not required. Out-in interactions are enterprise application agnostic, making it possible to avoid provisioning multi-party interactions using point-to-point application integrations.

2. ARCHITECTURE IMPLICATIONS OF OUT-IN

The conjecture that *the in-out pov will not scale to meet out-in objectives* is important to consider as we move toward development of a service-oriented architecture for Web Services. It underscores the thesis that *Web Services represents a significant technology discontinuity, not simply a trend that can be explained in terms of technology that we currently have and use*. The concepts of unit of work, work granularity, and orchestration (which we have collectively labeled UoW) provide us with a means to evaluate in-out and out-in povs on Web Services.

The table below draws from discussion given in the previous section to compare and contrast the in-out and out-in povs:

Table 1 -- Differences between In-Out and Out-In

| | In-Out | Out-In |
|--|--------|--------|
|--|--------|--------|

| | | |
|-----------------------------|---|---|
| <p>UoW context</p> | <p>The UoW is connection oriented. The in-out pov forces participants to build point-to-point application integrations or to adopt canonical process and data models in order to participate in multi-party interactions.</p> | <p>The out-in pov stipulates that there must be agreement upon the outcome of an interaction, but not an agreement on the process to realize that outcome. In this sense, the out-in UoW is business context and oriented.</p> |
| <p>UoW point of control</p> | <p>There is usually only one TP Monitor that manages a single system of transactional resources.</p> | <p>There is no single point of control through which interactions may be coordinated. The locus of control moves from a closed enterprise, having a CIO as final authority, to a managed, distributed business exchange through which business interactions are enabled as services.</p> |
| <p>UoW duration</p> | <p>Transactions are short-lived so that locking of critical resources is minimized.</p> | <p>Interactions are long-lived and inherently involve multiple parties. A technology fabric weaving business processes, services, documents and people seamlessly together is required.</p> <p>Focus must shift from OLTP to long-lived, loosely coupled atomic interactions with compensatory semantics. This means that our understanding of performance, scalability, and other attributes we mellifluously label “ilities” will necessarily change.</p> <p>Locking critical resources for the duration of a long-lived UoW would be unacceptable.</p> |

| | | |
|--------------------------------|--|--|
| <p>UoW structure</p> | <p>The structure of a transaction (its nesting, its ordering of activities, its exception and failure handling) is a direct function of enterprise application and database specifics.</p> <p>UoWs are fine-grained and contain activities expressed using application and/or database programming interface levels.</p> | <p>All participants in a multi-party interaction do not use the same application and technology infrastructure – so it is unreasonable to assume that a single transaction could be pre-structured to properly function across all participants’ infrastructure.</p> <p>UoWs are business process focused and are designed to exchange information in the form of documents that represent the type of information that human beings would exchange if manually executing the business process.</p> |
| <p>ACID restrictions</p> | <p>The TP Monitor is dependent upon knowing server system specifics so that it can implement ACID (e.g. it must be able to detect transactional resource failures in order to support durability).</p> <p>All resources that are to be transactionally managed must be XA-compliant.</p> <p>Commit/rollback is managed in short time periods using 2-phase commit over XA-compliant resources.</p> | <p>Enterprises will not expose their infrastructure to be managed by someone outside of their enterprise boundaries. The in-out form of ACID cannot be implemented in an out-in context.</p> <p>Some resources to be managed in an out-in UoW may not be XA-compliant, so all must be assumed to not be.</p> <p>Commit/rollback is managed in long time periods over non XA-compliant resources, implying the need for compensation and human mediation.</p> |
| <p>TP Monitor restrictions</p> | <p>There usually are restrictions placed upon the implementation of a TP Monitor that are programming interface (thus programming language) specific. In the J2EE world, this interface is known as the Java Transaction API, or JTA. Conformance to this interface enables an interface-compliant TP Monitor of choice to be plugged into a J2EE application server.</p> | <p>There is no accepted programming language or programming paradigm used to coordinate business interactions in and between enterprises.</p> <p>Web Service specifications (e.g. WS-Transaction and related specifications) do not make provision for TP Monitor access outside of an enterprise boundary. Instead, they stipulate that compensational semantics must be supported.</p> |
| <p>Exception management</p> | <p>Transaction exception handling is mostly handled in code. It is infrequent that a human would ever be involved in transaction exception management since this suggests critical enterprise application resources could be kept locked for unacceptably long periods of time.</p> | <p>Exception management will be handled both by humans and by software. The duration of an interaction and the fact that human beings will be involved in resolving exceptions make it impossible to use traditional enterprise infrastructure and techniques to manage out-in UoW business errors and exceptions.</p> <p>Human beings can improvisationally participate in handling business exceptions, providing the opportunity to make corrections in line, potentially avoiding the need to rollback and restart interactions when failures occur.</p> <p>Improvisation enables loosely coupled organizations to learn by more quickly and</p> |

| | | |
|----------------------------------|--|---|
| | | effectively capturing and codifying domain knowledge needed to automate business interactions. |
| Development and Management Tools | Traditional IDEs will be extended to enable development of enterprise web services that integrate with workflow, application server, integration and TP Monitor infrastructure. Focus will remain on short-lived UoWs. | The kinds of design time and runtime services needed to support long-lived loosely coupled asynchronous transactions with compensatory semantics must dramatically expand. A (distributed) business level operating system that understands different kinds of UoWs, different levels of guaranteed performance and security, different levels of predictability and ways to manage latencies, different degrees of human mediation, and so on, will be needed. This, in turn, will necessitate a new class of development IDE. |

The differences between the two povs lead to a conclusion that an out-in pov cannot be provisioned with the kinds of in-out technology architectures we see in enterprises today, which implies the following:

- ❑ A new architecture will be required to manage an out-in UoW that supports: (1) a compensation-based commit protocol as a means to *roll back* the effects of interactions if they fail; (2) document centricity; and (3) a way to orchestrate interactions in a way that easily scales as interaction participants, with their unique business and data-related processes, come and go. This new architecture will impact the enterprise application market, the way vendors construct their application products, and the way that enterprises develop business applications in the future.
- ❑ A new technology fabric will be required to manage the web service-oriented composite application runtime.
- ❑ A new development environment will be required to develop web service-oriented composite applications and package them for deployment.
- ❑ Traditional IT norms will be challenged.

2.1 New Way to Architect

2.1.1 COMPENSATION AND EXCEPTION HANDLING

Atomic OLTP transactions are important building blocks for Web Service implementations within an enterprise, but they are insufficient for comprehensively managing and coordinating them because Web Services may be long- as well as short-lived. AND since some resources to be managed in an out-in UoW may never be XA-compliant, all must be assumed to not be. So an alternative to an entirely automated 2-phase XA-compliant rollback/commit must be provided to manage long-lived UoW errors and exceptions. This alternative, called *compensation*, represents the ability to *undo* selected side effects of a long-lived Web Service-based UoW when failure conditions occur so that the interaction state may be made consistent.

Web Service compensation plans will be implemented in the form of compensation *handlers*, which is similar to how exception and error management is implemented in 3GL/4GL programming languages (e.g. Java, C++ and SQL). However, unlike its programming language counterparts, a compensation handler

must be implementable in two ways: (1) as part of the interaction definition, and (2) improvisationally/dynamically.

Some business errors and exceptions can be anticipated at interaction design time, so it must be possible to develop *compensation handlers* when an interaction is defined⁴. But it would be fallacious to assume all business exceptions could be known in advance, so it is also important to be able to implement compensation plans and make corrections *in line* while an interaction is running. To support this more dynamic and improvisational form of compensation, it must be possible for both human beings and processes to browse and manipulate interaction state, business documents, and business error and exception conditions *as errors or exceptions occur*, rather than afterward.

In-out UoW error and exception handling *usually* occurs after an error or exception is detected and execution environment state is rolled back to what it was just before the UoW started. This type of exception handling would make exception management difficult to impossible for long-lived UoWs: it may not be possible to restore state to what it originally was before a long-lived UoW started. Enabling human beings to improvisationally participate in handling business exceptions provides the opportunity to make corrections in line, potentially avoiding the need to rollback and restart interactions when failures occur. And, in some sense, the ability to improvise allows the new seamless enterprise to *learn* by more quickly and effectively capturing and codifying domain knowledge needed to automate business interactions within a loosely coupled community of practice.

2.1.2 DOCUMENT CENTRICITY

Documents (e.g. purchase orders, mortgage and legal documents, faxes and financial trades) are used today in multi-party business interactions to capture interaction context/state and constraints on the interaction, and to represent business information exchanged between interaction participants when such interactions are conducted manually. It is this type of granularity and grouping of information to which we refer in out-in interactions when we use the term *interaction document*.

When in a Web Services context, XML/XSLT technologies are a natural fit for representing interaction documents because:

- ❑ They can be used to express the schema of an interaction;
- ❑ They can be used to express business information;
- ❑ They can be used to describe constraints on the interaction, and on business information;
- ❑ They can be extended with functionality implemented in a 3GL if necessary;
- ❑ XML namespaces provide a way to organize information and eliminate namespace collisions, making it possible to track document amendments as the interaction progresses through its life cycle; and
- ❑ The IT industry is galvanized around XML use for expressing information (e.g. Rosetta Net, OASIS, etc.).

By using XML technologies to express context/state and complete business information in an interaction, it becomes possible for a human participant to browse interaction information as easily as a computer. This makes it more practical to involve *both* human beings and system processes in an interaction. People can

⁴ Emerging standards for Web Services (e.g. BPEL4WS) recognize and support this.

see a complete set of information and both understand how they should function as a normal interaction participant, and as a mediator when problems occur.

Additionally, using *business information of the grouping and granularity that people usually process* as a heuristic for developing Web Services ultimately will result in a simpler Web Service API's and, consequently, simpler and possibly better performing supporting infrastructure (e.g. adapters with coarse-grained service-oriented API's that limit fine-grained information requests).

2.1.3 CONSTRAINT-BASED ORCHESTRATION

Constraints may be used to construct logical equivalents to the constructs of a procedural workflow language. While a computer scientist would probably feel a certain degree of discomfort over using non-deterministic technologies and techniques for business orchestration, there are assumptions that can be made on the use of constraints and constraint solvers that simplify and qualify their application to orchestration as follows:

- We are not likely to deal with systems of millions of constraints that require so much time to solve that use of constraints becomes impractical;
- It is possible to bound the constraint domain just as people do when they act with bounded rationality;
- There are reasonable and user-friendly ways to represent constraints to humans who will participate in interactions as information providers, as well as debuggers of interactions; and
- There are user-friendly ways to enable service interaction developers to map constraints to services.

Given these assumptions, constraints could provide a compact means to *script* interaction flow in a non-deterministic way that focuses considerably more upon information than it does application protocols. This enables scalability, extensibility and specialization beyond the type of point-to-point procedural scripting using directed graph workflow technologies like BPEL4WS, especially as the complexity of business interactions increases and as interaction participants come and go. Constraints even today are used to express business rules necessary to ensure the integrity of business information being manipulated in the interaction. And they can be used to support improvisational interfaces for human mediation when problems occur and exceptions are raised. Such an interface would be analogous to an advanced debugger for multi-party business interactions.

The orchestration component of a web services technology stack is the component that enables execution of composite web service-based applications. We use the term *interaction server* to refer to this component because it manages web services that map to the types of business processes and capabilities a business institution would use to implement and govern its business interactions both internally and externally. Essentially, an interaction server is a next generation application server that is entirely based upon and manages the life cycles of constraint-oriented composite web services.

Constraint orientation enables coordination that can be both statically and dynamically defined as part of an interaction definition. Constraint-based flow (as opposed to a directed graph defining one possible flow of perhaps many) enables an interaction to represent a family of workflows that produce a common business outcome in the form of a composite document. The capability to represent multiple flows simplifies and speeds the construction of web service-based interactions both inside and outside of an enterprise.

2.2 New Runtime Technology Fabric

As we consider the computing fabric that has become common place in enterprises today, we can quickly conclude that fabric components fall short of what is required to manage web service-based application systems:

- ❑ Security more often than not targets the enterprise as defined today. While security components make provision for authentication using credentials and provide the convenience of single sign-on, considerable work must still be done to address XML encryption/decryption as a function of roles played in a business interaction, protection against application denial of service attacks, and so forth.
- ❑ Policy-based service management for business as well as network and technology management and monitoring is not standard fare in enterprise management applications and infrastructure. At present, the ability to dynamically late-bind to various transports as a function of policy is uncommon if implemented at all.
- ❑ The ability to view the runtime state and context of work being performed and participate in controlled and improvisational ways to manage exceptions or mediate in some way presently is limited (consider a kind of runtime debugger as a desired endpoint).
- ❑ Application servers that are given responsibility to manage the life cycle of functional components and to administer the runtime services that these components require are specific to application technology, and were not designed to be used natively for web services. They were designed for use within the enterprise. Use of them outside of the enterprise inappropriately would bring with them the cares and complexities of the enterprise – there will be plenty of cares and concerns outside of the enterprise to be managed without borrowing from other places.
- ❑ Workflow engines not designed for use with web services frequently are designed to participate in enterprise units of work. In some cases, workflow products insist on playing the role of TP Monitor. This would be inappropriate outside of the enterprise where there is no single locus of control.

In addition to the deficiencies of enterprise fabric components, it is also important to note the emergence of hardware accelerators for XML processing (e.g. XML Transformation, XML Content Routing), XML Firewall and Security, and legacy system interoperability. While current day enterprise fabric components perform a subset of the functions now implemented in hardware, they were not necessarily designed to interoperate in a loosely coupled way with hardware innovations as these emerge into the market.

Further still, it is important to highlight the lack of support for monitoring and controlling business interactions at the business level that enterprise fabric components give. While it is possible to correlate business activities through log entry correlation, this does not mean that doing so is straightforward or easy to do. Some system management vendors are only now implementing limited forms of such functionality relative to common enterprise application technologies and applications today. Build-out of this functionality for use with web service-oriented platforms is work in progress.

Googling the Internet for vendors who implement fabric components, and who might have already implemented a full web services fabric provides a somewhat sobering (for CIOs) and/or exciting (for entrepreneurs) status report of where the market is today relative to a complete fabric for web services: many fabric components are emerging, but they have not yet been woven together.

2.3 New Development Environment

Web services change the way applications are developed. The more traditional type of application

developed with Java and .NET (component- and object-based), or COBOL and PL/1 are typically developed and their deployment centralized within an enterprise. A full-featured programming environment supports application development as a function of procedural and object-based components (known at design time) for deployment in specific technology stacks. Not so in a web service context.

Applications in a web service context are composite or aggregate services. Myriad technology stacks may be used to publish primitive web services for uses that span enterprise boundaries. While we are seeing BPEL development environments that incorporate XML-based transformation capabilities in them, it is important to note that there are plenty of other “application services” that must be provided in the runtime and made accessible at design/construction time in order for web service-oriented programming to become a reality. Such services include: thread management, common memory management, XA and compensation-based transaction management, the ability to manipulate information in a runtime context (is this some combination of XSLT and Xquery?), and so forth. There is need for tools and frameworks that can be used to expose web services using information and data sources that are not web service-enabled. And there is need for frameworks such as portal frameworks that can be used to consume web services and present the results of this consumption for rendering in the variety of browser, rich media, and mobile device contexts becoming commonplace in today’s global workplace.

Finally, as web service governance and policy management increase in their importance as web service use increases, a repository and tools will be needed to capture such information as the description of web services, their relationships with other web services, policies relating to them, performance characterizations of web services relative to specific policies, and details of contexts into which they have been deployed (with services to keep this information current).

2.4 Traditional IT “Norms” Will Be Challenged

With new ways to architect platforms and develop new applications, and with a new runtime technology stack will undoubtedly come new developments that will challenge the common wisdom of enterprise IT. Here are at least three:

- ❑ It is common for IT to identify applications within its infrastructure that serve as sources of record for specific information types (e.g. a CRM system services as the source of record for the definition of *customer* and all customer-related data). As composite service-oriented applications are constructed, the source of record is likely to move closer to the composite application to limit dependence on legacy applications and avoid the expenditure of time and resources needed to perform information-based integration and document construction.
- ❑ Use of *legacy technology stacks* to build both primitive and complex service-based applications will (have to) be minimized to ensure the expected returns on web service-related investments are realized. This means that a more stringent discipline will have to be put into place regarding the architecture and design of new applications.
- ❑ The definition of enterprise will be challenged to the core. It certainly is commonplace to find applications that authenticate against a database using some general username and password pair *simply because they exist within enterprise boundaries*. Equally common is the fact that, once authenticated, a user is authorized to see and do considerably more than if the user were not an employee. The risk of such loose security is “minimal” only because it is loose within enterprise firewall boundaries. Such practices must be fixed if the benefits of web services are to be realized.

3. ORGANIZATIONAL IMPLICATIONS OF OUT-IN

Taking an out-in pov on web services has organizational as well as technology impact. Because web service technology is an enabler of new business models, the impact on organization makes sense. We note that the impact can be seen in at least the following areas: (1) users must be viewed as *global citizens*

within a virtual enterprise, thereby minimizing the distinction between internal and external users; (2) IT must equip itself to implement new *service development and operational models*; and (3) web services provide a mechanism with which an organization can implement a technology life cycle management strategy.

3.1 Global Citizenry

Security is at the core of how users are managed in any information system. Most enterprises have some form of security – even if this is a firewall and simple database authentication mechanism. Larger enterprises typically use some form of LDAP Directory Service to manage users for email purposes, support single sign-on, etc. Equally safe to say is that most enterprises do not go much beyond this point. It would be very unusual to find enterprise security so built out that it is possible to restrict visibility to data to a sub-document level as a function of a user role, or to manage access control lists to a business functional level. Yet it is precisely the fact that security has not been built out to this degree that precludes a company from behaving globally. Citizens of a so-called global company are not global citizens at all. Instead, they are organized into regional groups – each with its own Microsoft Active Directory or regional LDAP directory – with permissions assigned at the regional level. Semantics of role definitions are inconsistent across the company – so it would be very challenging to establish uniform corporate role definitions and access control lists.

Taking an out-in pov on web services demands that an enterprise treat employees in the same fashion as an external partner: each is a person that can interact with or act on behalf of the enterprise as a function of the roles that the user can play. Roles determine security policies associated with business services that can be invoked, and information that can be seen and/or published. Roles may also determine quality of service-related policies that will govern business interactions as a function of specific roles. Infrastructure – like a global directory service – must be installed and maintained, and information must be secured to a business service level (including the information set that a business service manipulates).

What is interesting about web services is that they provide the means to take greater control of security to a business functional level. They name business service interfaces and provide a metadata description of the information set they manipulate. Assuming canonical process and information models (a big assumption, but achievable), governance policies (including security) can be applied at the web service level to make global citizenship an achievable goal.

3.2 New IT Development and Operational Models – the Benefit of Improved Governance

IT plays many roles in enterprises today. In some companies, IT equates to network, desktop, and business information system management – with separate Engineering, Support, and Service organizations that manage business relating to corporate software-based products and services. In other enterprises, IT organizations do all of the above.

What is commonplace in IT regardless of the organizational model in place is that infrastructure is not usually so well managed that actual returns on IT-related investment can be reconciled to projected returns, the business can feel confident about guaranteeing its partnering capabilities with service level agreements, and IT can justify standing against the business urge to cut IT budget when cost saving policies must be implemented.

Consider, for a moment, an IT organization – including network, desktop, and business information system management together with Engineering, Support, and Service functions relating to corporate software-based products and services – that runs itself as a software product and service provider *business*⁵. The word *business* suggests an operating model that requires the business to formalize its definition of products

⁵ Credit goes to Michael Chang of Nissan North America for taking this point of view to an IT operational model extreme.

and services that can be provided out to the desktop (e.g. the desktop itself, the network and phone system connections, software productivity applications and update service, security credential administration, provisioning of email, access to business critical services/applications and information sets, and product support/help desk). Assume the business model is based around the consumer as a service subscriber that receives a monthly invoice for each product/service used during the month, with rollup on a cost center basis. The business must be able to present a catalog of products and services that an employee requires/can afford to consume on a recurring basis; and it must be able to track product/service use for billing purposes. This requires IT development efforts to go well beyond its common stop point (minimal documentation, deployment only with the help of J2EE gods, can monitor use by sifting through log files – but this requires significant effort to routinely report business service-specific activities).

Whether IT formally bills cost centers in the form of some chargeback model or not, taking a service-oriented view of their business across all the functions that IT provisions establishes traceability to actual costs to develop and maintain products and services. This visibility identifies where IT infrastructure must be more rigorously managed to control expenses. It identifies where cost-cutting policies will have negative impact, and where it is justified. It identifies areas requiring investment. It identifies holes in funding strategies relating to the transfer of control to IT of products and services developed outside of IT. It justifies why IT should hold the business accountable for developing a complete product/service, including the operational hardening and build-out necessary to incorporate it as a well-formed business fabric component. In fact, it defines the core components of the corporate business service technology fabric, and essentially articulates the critical importance of a unified service and technology strategy across the company – whether actual or virtual.

Taking an out-in pov on web services suggests new IT development and operational models that are directly traceable to business value. It also packages business services in a way that enables the business to combine and recombine services and directly enables innovations in business practices and agility in business processes. Such packaging also permits incremental change (in the form of policies, or in the deprecation or replacement or addition of business services). In fact, an out-in pov provides a framework for sustained innovation in the face of changing business practices and models [5].

3.3 Technology Life Cycle Management Strategy

Technology life cycle management is difficult. It often comes in the form of a big bang replacement of a mission critical application or major infrastructure upgrade that never seems to cost less than or equal to initial estimates, never seems to be as “out of the box” as originally hoped, and never completes within a reasonable time distribution away from the original estimate – delivering to full expectations regarding functionality and quality. And it would be naïve to believe that simply adopting a web services pov would magically make technology life cycle management easy. It won't. BUT it does offer a strategy to manage it.

Third generation programming languages since the 1980s have included constructs for defining software interfaces separate from implementations of and compliance with these interfaces. To be fair, some of the rationale behind introduction of these interface constructs was to circumvent perceived limitations of object inheritance models. But another recognized benefit of interfaces was correctly noted to be the enablement to replace interface implementations without impacting code dependent upon having *an* implementation to use. Web services enable the same type of encapsulation at a technology level because they are technology stack agnostic. Furthermore, web service frameworks make it possible to introduce interceptors that can participate in web service invocations using a kind of software delegate pattern. Interceptors can be positioned before or after actual web service invocation as a means to implement policy enforcement, specialized logging and monitoring (including billing for web service use), and so forth.

The ability to be technology agnostic and the ability to incorporate interceptors surrounding web service invocation form the basis of a technology life cycle management strategy. Consider the following example as a simple explanation of this point.

Telecommunication companies often have many billing systems that do far more than the “billing system” name suggests. In fact, a billing system often is involved in product/service order management and provisioning, workforce management, call detail record rating and statement generation, and customer support. It is clear to see that such a system is very critical to the business – and no sane person would ever replace such a critical system in a big bang fashion. In fact, one can argue that the only rational way to replace such a system would be to buy or implement a functional replacement for the system, run the new system in parallel with the old system for some period of time (say between 1-2 years) to populate the new system with accurate information and verify rating and statement generation and other key functions are properly working, and then turn off the old system except for historical purposes.

How could this be accomplished with web services? One possibility could be the following:

- [1] Encapsulate existing key business functions as web services;
- [2] Develop new business services that corresponds to web services produced in [1] that “listen” as interceptors whenever business services in [1] are invoked;
- [3] Deploy web services in [2] as possible/sensible, compare information produced in [2] to information produced in [1] for validity;
- [4] Complete [2]-[3];
- [5] Run the business on business services in [1] until entirely confident that [2] is reliable and the business can be run on new business services;
- [6] Run the business on [2], making business services in [1] interceptors when [2] business services are invoked, and as a backup;
- [7] Decommission [1] and the corresponding legacy application (perhaps except for historical purposes).

To be fair, there are ugly details that cannot be seen in the process outlined above caused by the inability to be entirely technology agnostic in a technology migration like the one described here. But these details can be managed. And the end result of migrating to a set of business services exposed as web services is that subsequent life cycle management is likely to be considerably easier based upon the assumption that technology agnostic interfaces are now in place.

4. BENEFITS OF OUT-IN

Taking an out-in pov of web services implies a willingness to think and act in a globally enabled fashion. According to this pov:

- ❑ The distinction of internal vs. external users of business services is replaced with a global user concept supported with software and hardware infrastructure that secures the enterprise, whether virtual or physical, and enables role and policy-driven management of user interactions.
- ❑ Technology business services are well defined and technology agnostic.
- ❑ Business services are defined according to a common meta model such that it is possible to reconcile semantic impedance mismatches between partners using standardized infrastructure, though this may be semi-manually accomplished for some time yet.
- ❑ The ability to view business services as web services (homogeneous from a technology point of

view) supports on-going innovation in the face of changes to business practices and models – and such visibility provides opportunity to measure business performance and effectiveness as a function of configurable (possibly customer specific) policies.

Consequently, the out-in pov leads to managed collaboration between loosely coupled business models without inappropriately circumventing or being limited by necessary enterprise edge-level technologies. Because the technology heterogeneity problems are localizable, the pov enables enterprises to reconsider their boundaries – paving the way to form loosely coupled communities of practice more easily and on an as-needed basis. And while the coupling is loose, the ability to govern both design time and runtime use of business services suggests a level of robustness that exceeds many of the enterprise information systems in place today.

5. DETRIMENTS OF OUT-IN

To be certain, moving to any web services-oriented point of view will be painful. The web services community is hardly unified in its point of view of how web services should be implemented, what the important standards are, the urgency of defining stable/implementable/well formed (especially in an operational sense) standards, the definition of a complete web services technology stack, and so forth. Change at the technology level is and will continue to be the norm for quite some time. Change will certainly cost money and other resources in order to keep systems web services interoperable as web service standards and technologies mature. While the belief that legacy applications can provision the first generation of business services, the cost of discovering and surfacing web services using these applications will test the patience from the boardroom to the network operation center.

So where does a company start?

Probably the very best advice is to start at the edge of the business [6]. Before it makes sense to aggressively pursue a web services strategy, a business must understand what its business model is, and what its key business services are. Remember that these business services are not dependent upon the kinds of business applications run in the enterprise IT department – rather, the relationship *should* be the other way around. Perhaps this start point is a hint concerning which one of the two points of view discussed in this paper should be adopted.

6. SUMMARY

Web Services provide us with an opportunity to conduct business in ways that scale beyond our current notion of *business enterprise*. Because there are different points of view about how the potential of Web Services architecturally can be realized, we identified the key concepts of *unit of work*, *work granularity* and *orchestration* as useful when evaluating approaches to developing service-oriented architectures for Web Services. In particular, these concepts, which are abbreviated as *UoW*, were used to compare and contrast the *in-out* and *out-in* points of view on Web Service-oriented architectures to show: (a) while both points of view can lead to the development of *service-oriented architectures*, the architectures will be substantially different; and (b) the assumption that *current in-out enterprise architectures can be extended to realize out-in goals* may be misguided. (b), in particular, is a very controversial position to take. Equally polarizing is the belief that a new *out-in* architecture will result in the demise of new enterprise application development as we have come to know it.

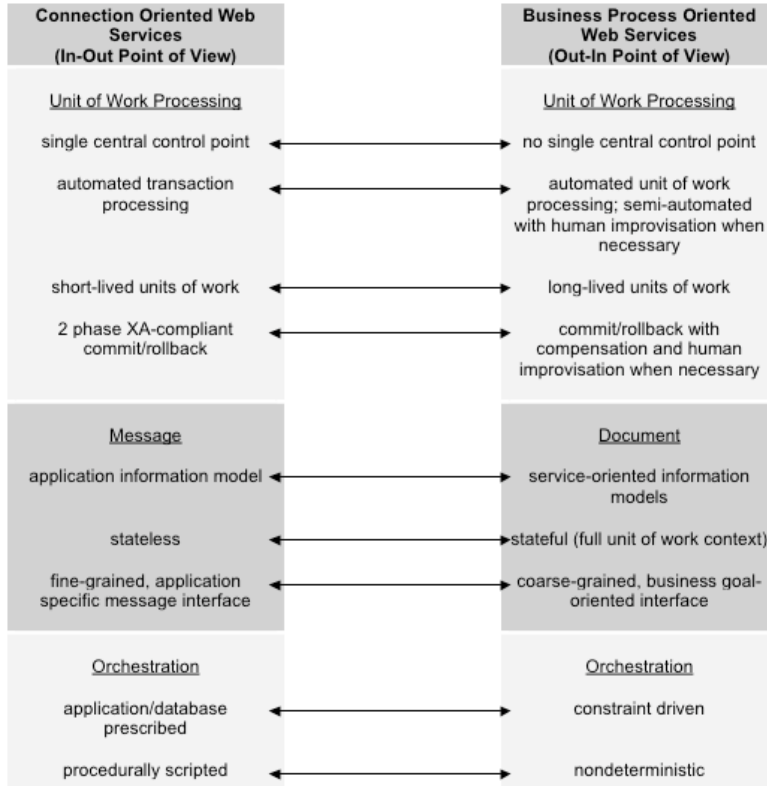


Figure 2

It is crucial to examine the key distinctions between the in-out and out-in povs, shown in the figure above, we begin to identify what is architecturally important when constructing web service-oriented platforms and applications, and how enterprises can leverage their current infrastructure to participate in multi-party business interactions. The out-in pov could lead us to an architecture that not only will generalize well as loosely coupled communities of practice join together to conduct business, but also will specialize *into* the enterprise in a way that may lead to the refactoring and simplification of current enterprise architectures.

7. BIBLIOGRAPHY

- [1] Hagel III, J. and J.S. Brown, Service Grids: The Missing Layer in Web Services. Release 1.0. Edventure Holdings, Inc., New York, Dec 23, 2002, 20:II:I-32.
- [2] Perry, D. and A. Wolf, Foundations for the Study of Software Architectures. ACM SIGSOFT Software Engineering Notes, 1992.
- [3] <http://www.opengroup.org/sib/details.tpl?id=C193>, Apr 2005.
- [4] Carr, N., IT Doesn't Matter, Harvard Business Review, May 2003.
- [5] Does IT Matter? An HBR Debate. Harvard Business Review, June 2003.
- [6] Hagel III, J. and J.S. Brown, Break on Through to the Other Side: A Missing Link in Redefining the Enterprise. http://www.johnhagel.com/paper_breakonthrough.pdf, 2002.

8. ACKNOWLEDGEMENTS

An earlier version of this paper was developed in 4Q2003 in collaboration between Tom Winans, John Seely Brown and Nancy Martin. This version updates the thinking set down then based on research in the web services marketplace from 4Q2003 to 1Q2005.

Thanks are given to Stan Raatz, Ken West, Roman Stanek, Radovan Janecek and Adam Blum for their insights and discussion relating to the development of the first version of this paper.

9. AUTHORS



Tom Winans is an IT management and technology consultant. He assists clients with pre-investment technology diligence, post investment portfolio management, development and implementation of organizational and software production methodologies, distributed service-based application architectures, and performance of project triage.

Tom has worked and consulted in the telecommunication, manufacturing, and IT industries for over 20 years. Tom can be reached through his web site: <http://www.concentrum.com>.



Dr. John Seely Brown (JSB) formerly served as the Chief Scientist of Xerox Corporation and the Director of its Palo Alto Research Center (PARC). He was deeply involved in the management of radical innovation and in the formation of corporate strategy and strategic positioning of Xerox as The Document Company.

Today, JSB serves as “Chief of Confusion”, helping people ask the right questions, trying to make a difference through his work- speaking, writing and teaching. He can be reached through his web site: <http://www.johnseelybrown.com>.