Alternative Technologies

Embracing SOA

The Benefits of Integration Independence

David McGoveran
Alternative Technologies
6221A Graham Hill Road, Suite #8001
Felton, California 95018
Website: www.AlternativeTech.com
Email: mcgoveran@AlternativeTech.com
Telephone: 831/338-4621 Facsimile: 831/338-3113

Report Number 20060125

Research Co-sponsored by TIBCO Software, Inc

Disclaimer and Notice

This report is produced and published by Alternative Technologies (AT hereinafter), an independent consulting and analyst firm in Boulder Creek and Felton, CA. The information and opinions presented in this report are exclusively those of Alternative Technologies, except where explicitly quoted and referenced. Although reasonable attempts have been made to insure the accuracy of the report, no guarantees or warrantees of correctness are made, either express or implied. Readers are encouraged to verify the opinions stated herein through their own efforts.

This report is based on research co-sponsored by TIBCO Software, Incorporated (TIBCO). TIBCO is granted a non-exclusive license for unlimited distribution of this report in its unabridged and unaltered form. Neither this report, nor any abridgement or derivation of this report, may be otherwise reproduced in any form or media without the explicit written permission of AT. This report is, in general, the sole property of AT and protected under International Copyright laws.

For information about this or other reports, or other products and services (including consulting and educational seminars), contact Alternative Technologies directly by telephone, mail, or via our Web site:

Alternative Technologies
6221-A Graham Hill Road, Suite #8001
Felton, California 95018
Telephone: 831/338-4621 FAX: 831/338-3113
Email: mcgoveran@AlternativeTech.com
Website: www.AlternativeTech.com

TABLE OF CONTENTS

1. Introduction

The history of IT has been, in one sense, a continuing struggle to computerize ever-increasing numbers of business functions in the face of change and limited budgets. Of necessity, the process began by creating islands of automation in which only mission critical business functions were computerized. Historically, the need to move data between systems was realized early on, and ultimately led to data integration strategies. Attempts to eliminate redundant data entry and the many errors it inherently created ultimately led to application integration efforts. From a methodological perspective, the need for integration can be understood as a failure to properly address present and future requirements. Over the last thirty years, technologies, methods, and standards have been developed with the sole purpose of addressing the "integration challenge."

Integration has become a discipline in its own right, and is significantly different from those technologies, methods, and standards used for application design, development, deployment, and maintenance. These differences have resulted in a fragmentation of IT into two types of teams and resources – those focused on traditional IT development and maintenance and those focused on the integration challenge. The consequences of this "house divided" have been far reaching.

In this report, we examine some of the factors that drive the need for integration. The complexity of the integration challenge is increasing much faster than the ability to integrate and it is necessary to gain control over these factors at their source. Several approaches to the integration challenge are discussed, and the value of a SOA as a solution is explained.

In summary:

- integration has always been, and will remain, an inherent component of IT practice
- an SOA approach attacks the root causes of the integration challenge, enabling IT alignment with business
- service reuse must be more than code cloning, and business services must be distinct from technical services
- SOA provides an opportunity to embrace integration so that it is a natural consequence of design/development, reducing the communication gap between business and IT
- SOA should not be constrained by the limitations of the Web Services and BPEL standards
- the natural heterogeneity of IT environments is best addressed by a platform independent approach to SOA, referred to here as "integration independence"

2. The Root Causes of The Integration Challenge

Integration is a strategic IT requirement that cannot be eliminated. The necessity of integration is traceable to at least two root causes. First, IT is under tremendous pressure to deliver tangible business value and not just technology for technology's sake. In attempting to align with business goals, IT must struggle constantly to determine business requirements and translate them into appropriate technical solutions. Second, IT departments, just like most other departments, are under pressure to preserve and leverage assets, as a way of reducing costs. A more detailed analysis of these root causes will provide guidance in addressing the integration challenge.

A Requirements Communication Gap

In computing's early decades, IT was a scarce, extremely expensive resource. IT's use was effectively restricted to highly repeatable business activities triggered by predictable business events. Considerable care was taken to drive IT efforts from business requirements, encouraging cost effective IT use. This was achieved with some efficacy by IT business analysts, a profession for which one could be trained and employed well into the 1980s. IT business analysts were the intermediaries between users and developers, who necessarily speak different languages, use different tools, and have different perspectives. Their job was to understand both the business and technology, and translate business requirements into IT requirements, bridging an inherent *requirements communications gap*. In this context, IT requirements include both application-specific functional requirements and longer-term strategic requirements. The latter enabled IT to maintain a more integrated environment and a consistent architecture, at a cost of longer delivery cycles than are acceptable today.

Over time, the requirements communication gap between the users and developers perception of software has broadened. On the one hand, graphical user interfaces made applications superficially easy to use and understand. On the other hand, both application software technology and the methodologies used by IT business analysts became increasingly complex and obscure. The result? Misalignment!

The likelihood of poorly or incompletely communicated business requirements increases, further aggravating business-IT misalignment.

The requirements communication gap has had two components, only one of which was recognized (albeit inaccurately). The first gap component was a "post office syndrome" – as the number of transitions involved in gathering and communicating requirements increases, the number of possible mistranslations grows combinatorially. The need to reduce communication barriers helped fuel the fervor for prototyping, object oriented, and model-driven software development approaches. These approaches certainly reduce development time and generally lead to greater satisfaction among users. Unfortunately, they also eliminate the IT business analyst role without conveying the IT business analyst's knowledge of the business or of interviewing subtleties to the software designers and developers now involved in requirements

gathering. One result: Strategically important improvement opportunities often go unrecognized and unaddressed by IT during iterative prototyping.

The second gap component is more subtle than the first, arising because technologists with little business experience too often attempt to gather requirements directly from business users. Although senior business and IT management consider budgeting and authorization, they generally step aside for requirements gathering. In the early days of IT, this was reasonable. Software was used to automate highly repeatable operational activities and perceived as having little to offer strategic business activities. Indeed, the role of IT in managerial decision support was minimal. All this has changed: Software is more sophisticated and can now not only support, but strongly affect, strategic business activities.

Without IT, businesses have little hope of maintaining the frantic pace demanded by business issues such as regulatory compliance, global competition, real-time enterprise, and on and on.

But if IT fails to address the integration challenge, IT becomes the source of drag on the business, and a serious impediment to business change and responsiveness.

Effective IT Asset Management

Were every application developed anticipating every possible interface (and, obviously, it is not), integration would be trivial. Even if this were achieved with all future applications, integration with previously developed applications would still be necessary. Ideally, a business could address the problem of integration by starting fresh: implementing the correct software engineering methodology, an open or complete infrastructure, toss non-compliant systems, buy the necessary software, and be done with it. This is, of course, pure fantasy for any business except a few startups. In general, businesses cannot afford to retire all IT assets and start fresh, but must leverage most of what is already operational.

The enduring question is how to fix an existing integration problem that is compounded almost daily. As will be seen, this is essentially a problem of IT asset management, which has two key components. First, the costs of asset retirement and replacement must be minimized while maximizing the benefits. Second, asset utilization must be optimized until retirement and replacement are appropriate. To put it another way, IT needs to be able to preserve and leverage existing assets.

IT asset retirement and replacement is fraught with risks and costs, and must be managed carefully over time. Existing software systems are coupled to and depend on both business operations and IT assets in surprising, and sometimes obscure, ways. An important example of the consequences is disruption of operations. Every IT manager knows that replacing any software system is likely to disrupt operations. Operations are also disrupted whenever new functionality is introduced, even though that functionality may be highly desired by business users and deemed necessary to future operations.

Change of any kind is inherently disruptive. However, managed disruption can have a net desirable impact. In particular, the disruption should be controlled so that its dominant affect is

limited to undesirable activities or processes, thereby discouraging or eliminating them. That is, a change in IT assets – whether deployment of a new application, enhancement of application functionality, or application integration – is acceptable only when it disrupts, disables, or replaces a bad business process or operation and *simultaneously* enables a better business process or operation. This result can only be achieved consistently in one way:

IT must drive change top-down, directly from business process and operations into the IT infrastructure.

Overriding almost every item on IT's agenda is the goal of preserving and leveraging earlier investments, known formally as optimizing asset utilization. This goal has led to the adoption of most changes to software engineering methodologies, from structured design that sought to optimize time and resources by reducing errors, to object orientation that sought to optimize code reuse. These agendas were marginally successful, largely because they improperly defined asset utilization to meet technical goals. All too often, IT assets are forced into premature retirement and replacement for obscure, tactical, or arbitrary technical reasons. The cumulative strategic costs for such "rip-and-replace" policies are exorbitant.

IT must find a way to preserve and leverage assets for as long as those assets deliver a net business value over the benefits of replacement.

3. The SOA Solution: Aligning IT with the Business

Enabling the Business Operating Environment

Existing IT infrastructures and how businesses establish IT requirements are both mired unnecessarily in costs that result from a high inertia past that has evolved badly. Business' window of opportunity is now too short to reintroduce a long, costly, error prone analysis and design phase. IT must somehow support strategic business decisions and requirements faithfully (without introducing technological buckshot) and in near real-time. SOA (service oriented architecture) and support for business process management, business activity monitoring, and complex event processing, *properly used*, is proving to be an answer.

SOA can give business direct control over the automation of many strategic business activities by eliminating much of the communications gap. This aligns IT more closely with business.

SOA has a long and honored heritage stemming from the earliest days of distributed computing, including simple two-tier client-server (with a single server), through three tier application server architectures and message-oriented middleware, to the multi-tier, service orchestration, and event-driven infrastructures. This evolution has been merely a technical and tactical response to IT's incremental challenges. As it crystallizes into a clear agenda, SOA offers the potential of a strategic, business driven response.

To be valuable to business users, a SOA must be presented and implemented with support for rapid delivery of business functionality as its most prevalent feature. Figure 1 shows a business functional perspective of the BOE (Business Operating Environment) that SOA enables. It emphasizes how business users can interact with a SOA by specifying, operating, and maintaining the business processes, business events, and business goals in the light of business performance monitoring, These must define services, service composition, and service orchestrations. Such a SOA leaves IT to provide technical services, monitor SLAs (service level agreements), and provide and manage the necessary infrastructure resources. The BOE separates business and IT functional roles without depriving IT of access to business requirements and goals. It permits IT to isolate business users from the inherent limitations and complexities of any particular technology. Simultaneously, the tangible effects of IT can be conveniently thought of as business services.

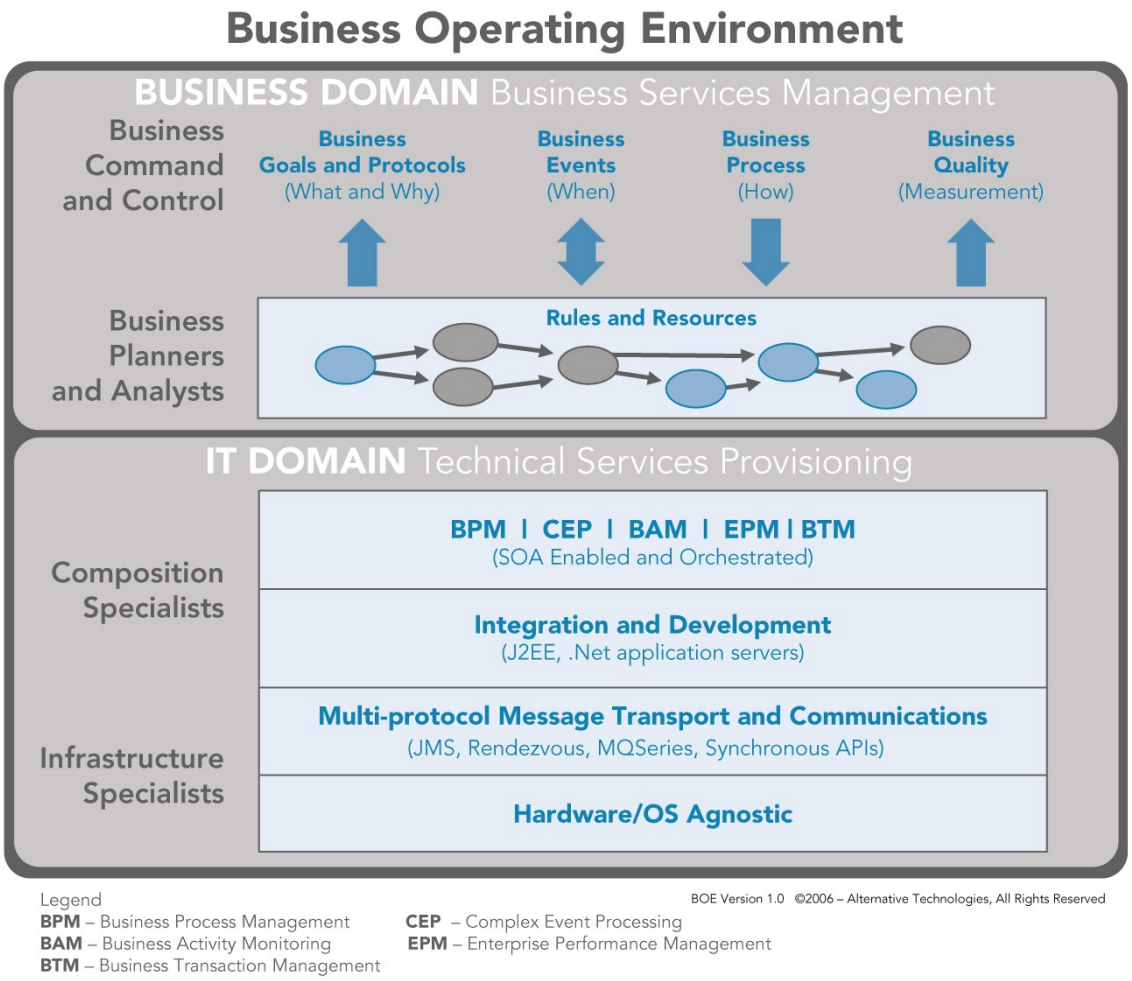

Figure 1

Services: Addressing the Gap

Many IT organizations find the process of implementing a SOA difficult to begin, in part because it requires thinking of the business as a collection of interdependent services. Traditional functional organization may obscure services, making them difficult to recognize and leverage, let alone computerize. Focusing on services, and understanding their nature, is a good place to start.

A service is performed anytime someone or something produces an output of some sort, whether it is material (a product, a component, or a supply), informational, or managerial. Services are always consumers as well as producers. At the edges of an organization services may appear to be pure producers (e.g., supply chain partners) or pure consumers (e.g., customers). Recognizing the fallacy of this view leads to supply chain management and customer relationship management. For example, and simplistically, if the customer is understood as a service that consumes goods and services and produces revenue, the optimization of that service clearly requires an understanding of the events, processes, goals, and activities that define the service interface.

Services should be classified into business services and technical services. Business services provide a function that is entirely understandable from a business perspective, in effect encapsulating or hiding the details of its implementation. The service definition depends on the business context, goals, and operational standards, but should not depend on the technology that is used to implement them. Alternative Technologies has long maintained that IT should be a provider of business services to its business clients, encapsulating IT resources. By definition, such business services deliver value directly relating to the business' primary purpose and can be understood and used without knowledge of IT artifacts.

Business users should not have to care how business services are implemented technically, so long as they faithfully address business requirements.

By contrast with business services, technical services expose the technical functionality necessary to accomplish business services in the context of available technical resources. They provide access to IT resources, and are used by developers in composing business services. A technical service should form a scalable, robust service abstraction over technical resources so that they can be managed on the basis of capacity requirements and technical innovation. Sufficient reason for separating business services from technical services is that technical services can change independent of, and at different rates than, business requirements.

SOA has the potential for being the main conduit for both of those service types. First generation SOA efforts have focused almost entirely on technical services rather than business services. The W3C standards based approach to service implementation, known as Web Services, has done much to give SOA credibility, but architects and developers should keep in mind that they are merely technical caricatures of business process, business transactions, and the like, which must be composed into their business counterparts. Treating Web Services as the *sine qua non* of SOA or BPEL (BPEL4WS) as the *sine qua non* of orchestration are hardly defensible on the basis of business requirements. There are, in fact, good technical reasons not to implement every service as a Web Service.

Few technical standards are motivated by business requirements (read Web Services standards if you doubt), being driven heavily by technical and vendor agendas. Nonetheless, advocates repeatedly – sometimes intentionally – confuse business terms with recently invented technical terms. Business event, transaction, activity, and process (and its variants such as orchestration, choreography, and coordination,) as used in Web Services carry tremendous, constraining technological baggage not assumed in the corresponding business terms. For example, although a standard like BPEL is certainly an important step in supporting service orchestration, it is hardly a tool for executing "business processes" – at least as any business manager would understand the term. Instead, it forces the real-world business process to be modeled in well-structured computing constructs that can be guaranteed to have a deterministic result, largely ignores human-based activities (as well as their scheduling and management), and is largely oblivious to business transaction requirements. These limitations alone make it unsuitable as the sole expression of service orchestration in composing business applications that reflect a business' operations.

Leveraging IT Assets

Services orientation and, more specifically, a SOA, can provide a mechanism and method for defining IT asset utilization to meet business goals. Every technical service can then be implemented in response to the requirement for a business service the performance of which is measurable by business costs and benefits. The resulting alignment between business and IT has obvious benefits.

If asset utilization is to be optimized, services must be designed for easy use and reuse. There are several requirements for such services, the most important being designing the service to expose only its primary function and for the most general use of that function.

Done well, enabling service reuse reduces the costs of development and integration, accelerates application development and deployment (i.e., agility), and reduces risk.

Reducing IT Costs

A second goal on IT's agenda is to manage – and typically this means "reduce" – anticipated future costs. The constant whipsawing effect of radically new technologies, standards, acquisitions, mergers, divestitures, regulations, market changes, and so on, are unlikely to be abated. However, the cost of responding can be reduced through a SOA. Technically, a SOA addresses many of the difficulties of building scalable, robust, distributed applications, and provides a framework for composing applications from reusable services.

Ensuring that newly acquired or developed software assets deliver business value – what we might call the "development challenge" – is only one half of the challenges IT faces. Having acquired and developed useful software assets, the useful life of assets and their value should be preserved and leveraged: This asset utilization goal is the essence of the integration challenge. It is instructive to think of integration as composition, whether applied to traditional software components (modules, function libraries, and so on), objects, services, or applications. Traditionally, IT activities were divided into development and maintenance. By contrast, the services-oriented perspective divides IT activities into services development, services

composition, and services deployment. To take advantage of this perspective, services must be composable with other services and sharable (i.e., they must be reusable). Traditional IT maintenance activities often involve services development, recomposition (changing which services have been composed or how they are composed), and redeployment (changing the services environment). In a nutshell, the latter two preserve and leverage the former through reuse, and by facilitating functional augmentation as a consequence of composition.

Integration as a necessary IT activity will never go away. However, it must cease to be a separate activity from development if the development and integration challenge are to be addressed. Properly used, a SOA attacks these challenges at their root causes. This is easily understood in the context of the three dominant root causes of integration: inherent complexity, unforeseeable requirements, and perpetual change.

- Inherent Complexity – IT has had the task of supporting business since its inception. Few business events, processes, or activities are simple enough to enable straightforward computerization. In fact, most are inherently complex and have resisted over fifty years of computerization. Furthermore, competition ensures that, although they may be compatible with industry standards or best practices, they will deviate to support competitive differentiation. By providing a uniform framework that exposes IT assets as reusable services, SOA provides an opportunity to manage inherent complexity as a natural and even desirable characteristic of IT.

- Unforeseeable Requirements – The need for integration is an artifact of unanticipated requirements. It is almost impossible for anyone to anticipate all business requirements for an application, or all the other applications and systems with which it will need to be integrated. As long as business processes can change (as they must if a business is not to stagnate and become non-competitive), so will the complex of applications necessary to implement them and hence, integration requirements will change. By enabling service composition and reuse, SOA provides the agility necessary to respond rapidly to unforeseen requirements.

- Perpetual Change – For a variety of reasons, the application mix is unlikely to be static. New technologies, business requirements (including compliance and regulatory requirements), business and technology standards, competitors, partners, suppliers, and partner processes, as well as mergers, acquisitions, and divestitures, each engender perpetual change. Those changes constantly create new integration challenges. SOA changes the nature of the integration challenge so that it is embraced as a primary activity within IT development and maintenance practices, rather than an after-the-fact and more costly response.

These root causes ensure that the world is not homogenous in terms of business, people, or technology, nor is it ever likely to be so. Despite the best efforts of standards bodies and the unrelenting effects of commoditization, diversity and conflict are necessary conditions for healthy, dynamic systems.

SOA enables all IT assets to be treated as services and both development and integration to become a services composition effort. Empowering IT to live with heterogeneity is a key reason to adopt a SOA.

To repeat, a SOA makes it possible to embrace integration as a first-class IT activity by completely merging it with development, rather than one to be avoided or eliminated. A SOA addresses complexity, unforeseeable requirements, and perpetual change by breaking functionality into reusable, readily understandable services, any combination of which can be predictably composed and reliably delivered.

By enabling heterogeneity and embracing integration rather than attempting to eliminate it, businesses are inevitably led to treat development and integration as synergistic activities within the IT process. This philosophy is essential to achieving the benefits of a SOA.

Service Reuse

The ability to enable service reuse is the key means by which a SOA supports both development and integration. Unlike services in a traditional application server architecture – which often depend on code cloning, services in a SOA are distributed and can be shared among many applications. To encourage reuse, it is not enough that services support standard invocation, orchestration or discovery: Services should also be designed to support a variety of deployment options. Many service deployment options are unrelated to its primary business or technical function, but are necessary for the use of the services in a particular deployment or application. Furthermore, many essential deployment options are services in their own right, but are used in one way or another by almost every service. Examples include service registration, data transformation, routing, security, transaction management, transport, and so on. These meta-services are properly part of the SOA infrastructure and should be dynamically configurable.

For example, although Web Services standards specify SOAP as the message exchange protocol, it permits the SOAP transport to be HTTP or JMS, and there is a good possibility that SOAP over UDP (i.e., for multi-cast) will be possible as well. Which of these should be used depends on the particular application and the environment in which it is deployed.

SOAP/HTTP is adequate for mobile and web-based applications where message delivery is less critical, while SOAP/JMS is preferable when robustness or transactionality is an issue. On the other hand, SOAP/HTTP can provide greater interoperability in non-Java environments.

In addition to these architecture and design considerations, encouraging service reuse means planning for ongoing maintenance that goes beyond adding new functionality to an application or system. Making services integration-friendly certainly requires that services be composable into applications and subsequently recomposable into new applications. This capability is, of course, the purpose of the many Web Services standards comprising orchestration, transport, discovery, and so on. In addition, however, services need to be refactored from time-to-time to encompass more general functionality (e.g., by introducing polymorphism). Put simply,

refactoring is changing source code without altering its interfaces or contracts. Because new functionality and technical resources are inevitable, some new version of a service will be needed. If services are truly distributed and shared, new versions will be deployable without restarting applications or processes. The alternative to refactoring is unnecessary service proliferation, which clearly discourages reuse.

4. Approaches And Failure Modes

There are four popular strategies for SOA adoption: single vendor, standardized platform or infrastructure, best-of-breed, and independence (or heterogeneity). Each of these approaches has strengths and weaknesses, and conforms to reality in differing degrees.

Single Vendor

Adopting the development and integration platform of a single vendor is attractive in a number of ways. First, committing to the solution provided by a single vendor eliminates the costs of evaluating multiple solutions and of managing multiple relationships. Decreasing the number of vendors involved in your infrastructure decreases the number of contracts to negotiate and purchase orders to process. Overhead for support and training decrease, and there is less question as to who is responsible for support. Rather than needing an in-house integration team to integrate components of an architecture, the customer can rely on the services of either the vendor or its system integrator partners. When a careful analysis of the vendor's vision (as evidenced by a clear and technically convincing roadmap), stability, proffered business functionality, and technical capabilities demonstrates overwhelming compatibility with customer requirements, a single vendor approach may be appropriate.

The degree to which the vendor's platform is standards-based or proprietary can vary considerably, in subtle ways, and with it the degree of vendor "lock-in."

In principal, the integration effort should be less complex and reasonably complete. The reality is that no vendor can possibly provide an off-the-shelf solution for every integration challenge, nor is it likely that any single vendor will provide the best solution for every customer. Vendors that provide so-called complete platforms often have another agenda such as selling enterprise applications, operating systems, or system integration services, so that the platform solution may be as focused on promoting other products and services as on solving the problem of asset utilization and integration. Furthermore, no vendor can foresee the future with respect to new technologies and standards, and, at best, continually attempts to keep its products up-to-date with new features and functionality.

The speed of introduction is always limited by factors such as the requirement of compatibility with existing customers' versions, vendor resources, the functional modularity of the product, and product uniformity (number of "specials"). Perhaps most significant is the fact that it is unlikely that a vendor's vision of IT will match the vision and requirements of every customer, especially given that fact that those customers need to be differentiated within their markets and are likely to undergo independent changes of marketing and technical direction. The

vendor must simultaneously be large (for long-term financial stability and breadth of experience) and be able to deliver customized solutions rather than a least common denominator solution. As the customer undergoes mergers, acquisitions, and divestitures, the vendor must be able to adapt rapidly to address an extremely broad range of IT environments. Obviously, the difficulty of achieving and maintaining a high degree of compatibility with a vendor is very high, making a single vendor approach a risky choice.

Standards-Based Infrastructure

Adopting a standards-based infrastructure is one way to avoid lock-in to a proprietary infrastructure and maintain a degree of control over the resulting implementation.

An effective method for adopting a standards-based infrastructure is to start with a general, vendor-independent SOA template. Refine it top-down based solely on functional requirements. Finally, select and assign standards to the protocols, interfaces, and services based on their ability to address those functional requirements.

Following this method, product acquisitions can then be seen as "filling in the boxes" in the architecture, and may involve any number of vendors.

Note that this approach is distinct from either selecting a single vendor merely because of product compliance with selected standards or adopting the reference architecture of a particular standards body. Both lead to a form of "lock-in" and are unlikely to be ideal for any specific business. IT has tried to institute standards-based infrastructures for many years. Standards can obviously contribute to lower design, development, deployment, maintenance, and integration costs. IT training and support costs are arguably decreased. When a business has the luxury of funding wide-spread and tolerant adoption of evolving standards, commitment to a standards-based infrastructure approach may be appropriate.

Unfortunately, adopting infrastructure technology on the basis of standards compliance requires a costly, up-front investment that is extremely difficult to justify on the basis of any measurable business benefit. Both software technologies and standards evolve, sometimes diverging. As a case in point, consider the now abandoned push to adopt CORBA for distributed applications. The focus on distributed objects that drove CORBA a decade ago is similar to the pervasive interest in Web Services technologies today.

An enterprise infrastructure involves such a large number of essential technical functions, components, layers, and subsystems that it is unlikely that any suite of standards can ever specify all of them adequately, let alone before technology trends change. Adding to this the time that it takes for vendors to adopt and implement all of the standards, it becomes readily apparent that a standards-based enterprise infrastructure is forever incomplete. As technologies and implemented standards diverge, the achievable business value erodes. All in all, a standards-based enterprise infrastructure can easily devolve into a least common denominator approach, so that few, if any, components are ideal for a particular business. Despite these weaknesses, adopting a standards-based enterprise infrastructure has benefits. However, a proliferation of standards or an infrastructure that is inappropriately complex in contrast to the needs of the business can defeat these benefits. To achieve the most benefit from a standards-based infrastructure, the realistic view that standards are evolving goals is essential.

Best Of Breed

A best-of-breed approach to infrastructure requires selecting each component on the basis of its functionality, rather than on platform cohesiveness. Like the standards-based approach, careful evaluation of components is required. These components are then integrated to achieve an infrastructure, the overall custom functionality of which is intended to best meet requirements. This approach addresses the problem of components that are less than ideally suited to the business' needs. Additionally, not all components of any particular vendor's platform will be of the same maturity or sophistication nor meet the unique and complex requirements of a particular business. The best-of-breed approach entails a freedom-of-choice that encourages vendor competition and product development without excessive vendor "lock-in."

An independent integration vendor that commits to a best-of-breed approach will embrace a broad spectrum of integration requirements rather than push an idealized infrastructure with a potentially costly rip-and-replace strategy.

Best-of-breed approaches require some caution. Blindly combining best-of-breed components to build an enterprise infrastructure does not necessarily lead to ideal applications. Realistic and optimized selection of off-the-shelf components must take into account the interaction among the components, both their strengths and weaknesses, the costs of multi-component deployment and system management, vendor financial stability, and so on. A best-of-breed approach requires considerable investment in the technical evaluation process. Today's industry analysts rarely aid the effort in any significant way because they shy away from the politics and costs entailed in detailed technical product analysis. Nonetheless, the approach offers an escape from naïve adherence to a single standard approach, such as forcing every service to be a Web Service or every process to have a pure BPEL implementation. When a business has strong architectural design and integration experience, along with the time, funding, and knowledge necessary for detailed technical evaluations, a best-of breed approach may be appropriate.

In the next section, we'll examine the benefits of an approach that supports independence and heterogeneity without sacrificing either standards or best-of-breed component selection.

5. The Benefits of Integration Independence

As intimated earlier, an enterprise infrastructure must support a dynamic, ever-changing environment that is specific to the requirements of a given business. To be successful, it should not be captive to the whims, agendas, or even the best predictions of vendors and standards bodies. On the other hand, every business needs to be able to take advantage of available products, the latest technologies, and released standards in deploying an enterprise infrastructure. The typical IT environment is heterogeneous and so requires platform independence to support the broad requirements of development and integration. By platform in this context, we mean the combination of application server, transport (both message and synchronous communication), DBMS, development environment, and other key infrastructure components that are often pre-selected (or at least given strong preference) when one adopts the platform of a particular vendor.

Heterogeneity pervades, and is a precursor to, integration. Most IT environments incorporate products developed by many vendors and deployed incrementally, possibly over decades. Given the changes to software technology that have taken place over the last ten years, (let alone the last thirty), it is hardly surprising that these products were never designed for easy integration. Even if this were the only sense in which IT requirements are inherently heterogeneous, it would certainly be sufficient to argue for a platform independent approach to integration.

IT is faced with what we might call "requirement heterogeneity", meaning that requirements are far from uniform and sometimes even seem to be in conflict. For example, some data integration requirements demand batch operation while others demand real-time, event-driven operation. IT is often faced with the need for a broad spectrum of business and technical expertise among integration specialists.

Most businesses engage in new operations, offer new products and services, and enter new markets over time. As the business evolves, the operational requirements of applications change. If two applications need to be integrated, but were designed at different times to incompatible operational requirements, integration requires a mediation facility to reconcile the differences.

Such differences can be almost impossible for a single platform vendor to anticipate and even more difficult to address over a spectrum of customers in many markets. Resolving them demands a highly-distributed service architecture such as is found in a true SOA approach. Even though a server-centric architecture (such as those often found in application server and application server platform approaches) can be a good way to achieve enterprise uniformity, lower certain costs, and establish a degree of seamless architectural integration from the departmental to the corporate level, server-centric architectures are less desirable for achieving a true SOA.

New standards are certainly important to platform independence, although they are not sufficient. For example, JBI (Java Business Integration) enables a degree of interoperation among heterogeneous components of a SOA by supporting service-oriented integration in a Java environment. By supporting JBI, vendors move towards greater interoperability and thereby improve responsiveness to the inherent heterogeneity of IT environments while reducing risk and lock-in to a vendor or particular solution.

There are many benefits to selecting a vendor that actively supports a platform independent approach to help deploy a SOA, including:

- The customer, rather than the vendor, can choose a solution that is best suited to the customer's needs.

- The customer can selectively apply a best-of-breed approach to any component without worrying about the negative and constraining effects of infrastructure interdependencies.

- The customer can leverage off-the-shelf products where appropriate.

- The customer can use the products of other integration vendors.
- The customer can leave, layer, and leverage existing assets.
- The vendor is more likely to provide SOA mediation facilities that accommodate the heterogeneity that is inherent in the customer's environment.
- Organizational and political hurdles, inherent in executing enterprise-wide SOA initiatives, are more easily overcome.

Most integration vendors fail to give IT asset reuse the priority it requires. They suggest that a "leave and layer" approach to existing assets is supported, at least until an opportunity arises to "rip and replace."

*All too often, vendors inadvertently circumvent the "**3 Ls**": **leave, layer, and leverage**." Unless the vendor's only agenda is mutual success with the customer, it is unlikely that existing assets will or can be treated as first class citizens in the architecture.*

And unless existing assets are first class citizens, it is doubtful that those assets can be leveraged to the customer's best advantage or that development and integration can be fully merged.

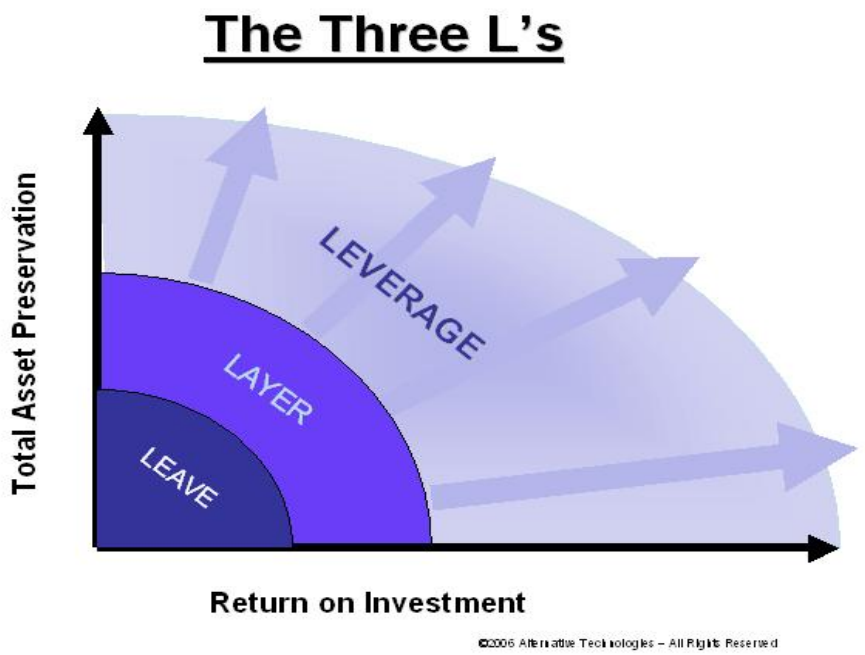

Figure 2

6. Achieving Integration Independence

The primary key to independence is selecting appropriate integration vendors. Vendors that can adequately support a platform independent approach to integration meet certain evaluation criteria. The weight each criterion is given depends on the business' particular requirements and the scope of the integration effort. In overview, the more important criteria are as follows:

- **Reliable, Flexible Infrastructure Core** – The vendor should offer an infrastructure core that is both reliable and flexible without vendor lock-in. The infrastructure core consists of message transport, service creation and deployment, service orchestration, and infrastructure monitoring and management facilities. The core should be platform independent.

- **No Dependencies –** A vendor's products should have no interdependencies. Product interdependencies are manifested in terms of functional restrictions when the products are not used together and coupled release schedules.

- **Existing Asset Support** – The vendor must treat existing assets as first class citizens in the customer's architecture. This means that the vendor's reference architecture directly incorporates existing assets and treats its own products merely as preferred tools for enabling the customer to implement a SOA.

- **No Agenda –** The vendor's only agenda should be their mutual success with the customer. Biases and a lack of sensitivity to integration complexities can be expected if the vendor uses the infrastructure to promote applications, particular standards, consulting services, etc.

From a technical perspective, a number of issues must be addressed to foster independence and support heterogeneity. The platform must be able to compensate for inherent differences in applications. Among the more important criteria for achieving independence are:

- **Sharable Services** – True SOA enables the development of services that can be shared among heterogeneous applications. While code cloning, as found in most application server implementations, is desirable, it is not sufficient to enable independence.

- **Model Independence** – SOA supports a model-driven approach to design, development, and integration. However, different applications and platforms use differing levels of description. This can greatly degrade the value of the model, and even make interoperation impossible. Platform independence requires the means to rationalize models so that they can be used across applications.

- **Interfaces and Protocols** – Defining and supporting standards is only a partial solution to disparate interfaces and protocols. Translation between disparate interfaces and protocols is, of course, essential as well.

- **Process Consolidation** – Heterogeneous applications and service orchestration platforms capture and implement processes differently. Platform independence requires a mechanism to consolidate multiple embedded processes and provide a single, combined view for management, monitoring, and administration.

- **Interaction Management** – Every application interacts with its environment in a particular way. Interaction timing requirements, levels of transactionality, error responses, and so on are all important. Rectifying interaction differences is properly handled in the infrastructure.

- **Architectural Independence** – Application architecture imposes constraints on many deployment options, interaction modes, invocation methods, etc. An infrastructure that promotes independence must enable integration of distributed, real-time architectures as well as server centric, monolithic, traditional mainframe batch, and other architectures.

- **Data vs. Message vs. Process Centricity** – Applications and platforms can be data-centric, message-centric, or process-centric, each with many possible implementations. The infrastructure should enable transformation among them if it is to foster platform independence. In this respect, keep in mind that Web Service interfaces are designed to support point-to-point connectivity rather than a general multi-point messaging infrastructure. As such, today's Web Services only emulate true publication and subscribe, which needs to be supported deep in the platform infrastructure.

We emphasize that the primary key to independence is selecting integration vendors that can adequately support a platform independent approach.

Combined, these vendor and technical criteria should enable the architect to establish a platform independent SOA and achieve the benefits of independence.

7. Conclusions

IT must reformulate its goals to survive, and software vendors better pay attention. Too often, and for a variety of reasons, software marketing focuses more on creating buzz than on conveying business value. A SOA's primary value must be more than solving a technical problem: It should achieve new business value by reducing IT costs such as TCO (total cost of ownership) and addressing business requirements directly.

With SOA, analysts are still needed, but – rather than IT business analysts accountable to IT – they can be business IT analysts accountable directly to the line of business they are servicing. Their job is to translate informal business requirements into business service specifications, (including operational interfaces; operational, procedural, timing, integrity, quality, and compliance rules; and dependent services/resources), that are precise and can be validated.

SOA enables changes to IT service provision to be driven top down and directly from business requirements. We can't really analyze and design to business requirements unless IT uses business language appropriately. But once it does, a SOA can transition beyond IT services and model-driven architectures to a Business Services *Orchestration* Architecture within a Business Operating Environment, with business as its user and IT as its administrator. Only by embracing integration as an intrinsic goal of the design, development, deployment, and maintenance life-cycle, refusing to foster more and more barriers between services and applications, can IT ultimately meet the integration challenge. And doing so will surely require accepting heterogeneity and demanding independence.

About the Author and Alternative Technologies

David McGoveran is the President and Founder of Alternative Technologies, an independent analyst and consulting firm founded in 1976. Mr. McGoveran has been a pioneer in the definition, technical architecture, and uses of Business Process Management Systems, having developed one of the earliest BPMSs in 1981 and defined the BPM marketing strategies and technical direction for companies such as Hewlett-Packard, IBM, Candle Corporation, and others. He has been lecturing and writing publicly on the topic of BPMS since 1997. He is Senior Technical Editor and co-founder of the eAI Journal (www.eaijournal.com), in which appears his monthly column *Enterprise Integrity*. Mr. McGoveran provides consulting and teaches seminars on Business Process Management, as well as other topics, helping users evaluate and select BPMS products.

Alternative Technologies publishes The BPMS Evaluation Scheme, a detailed analysis of the purpose, functions, and components of BPMS products. Analyses of popular BPMS products are also available.

For pricing and availability of these or other reports or other products and services (including consulting and educational seminars), please contact Alternative Technologies directly at (831) 338-4621 or via email addressed to mcgoveran@AlternativeTech.com.

Alternative Technologies
6221A Graham Hill Rd., Ste #8001
Felton, California 95018
Telephone: 831/338-4621 FAX: 831/338-3113
Email: mcgoveran@AlternativeTech.com
Website: www.AlternativeTech.com