CBDIReport

Business Flexibility Through SOA

An introduction to managing business flexibility using Service Oriented Architecture strategies and techniques

By David Sprott

ABSTRACT: Service Oriented Architecture can make a business more flexible, able to respond much faster to change. Of course flexibility is a very broad concept and the SOA can respond in many different ways. Demand for business flexibility therefore needs to be understood and managed in a systematic manner and treated as a functional or non functional requirement with appropriate business involvement. This report provides a framework for identifying and delivering the various types of business flexibility that SOA can deliver and shows how these may be achieved and measured. In the process the report provides clarity on what an SOA really is for a large enterprise.

Independent insight for Service Oriented Practice

Contents

This report has been commissioned by IBM Inc.

Introduction

It's a cliché but everyone agrees all enterprises are undergoing constant change. The problem is that change is a very broad concept and there are many types of change. So understanding how a business needs to be flexible to respond to many different types of change is a considerable challenge, not least because many types of change are extremely hard to predict.

Once upon a time change was widely assumed to be the product of management action, responding to new technologies, markets and opportunities. However in recent years we have all acquired a heightened awareness of requirements for change that are entirely outside of our control such as the risk due to terrorism and natural disasters that require us to be better prepared for the unexpected, as well as the requirements for statutory regulations which can have far reaching effects on business operations.

- **Speed of execution on strategy and innovation** identified as the key management challenge facing companies over the next five years. Fifty-four percent said their companies are shifting their focus to innovative business models rather than new products to achieve competitive advantage. *Economist Intelligence Unit 2005 survey of business leaders of more than 4,000 companies in 23 countries*
- **Since September 11th 2001,** it has become obvious to all that **the world is a risky place**. Even before that atrocity the world had seemed far from safe to many, especially those concerned with business and finance. The end of the dotcom craze and the bursting of the stockmarket bubble had already created huge uncertainty. But those are only the most recent examples of unexpected events that can make a mockery of people's plans. *The Economist Risk Survey 2004*
- **The (Basel compliance) rules are due to change** at the end of 2006. The changes are detailed, possibly far-reaching and certainly controversial. Despite regulators' attempts to deal with the objections, such a complicated set of rules is almost sure to throw up unintended consequences that no one has yet spotted. *The Economist International Banking Survey*

It is customary to attribute a large proportion of blame for lack of flexibility on the IT capability and organization, it must be said with some justification. In today's corporate and government environments there are very few business processes that are not completely dependent upon systems support. However that high level of dependence may be a strategic liability because the typical enterprise IT environment is incapable of rapid response to change. Most are the result of several decades of largely tactical, project related systems delivery activity and infrastructure investment built using a wide variety of incompatible development and operational technologies. The result is analogous to the multi-colored rock strata seen on an exposed mountainside, heavily compressed and tightly integrated.

Over the years conventional practice in systems investment has prioritized delivered functionality and speed of delivery and ignored full life cycle costs and the ability to respond to change. IT infrastructure is typically under utilized, slow and expensive to change with very high levels of human resource involvement. Consequently there is massive duplication of investment and effort and an extraordinary level of complexity combined with inadequate documentation and change processes. The result is a crisis in existing software investments because the typical response to change in major enterprises has not been improving in spite of real improvements in application delivery and management technologies.

It is natural and reasonable to look to information technologies to solve this crisis of inflexibility. For many enterprises application integration has been a temporary fix. However, integration

based strategies are only a short term solution because they generally create more dependencies and hence complexity and reduce future flexibility.

For some years it has been obvious that the way out of this mess is to introduce approaches based on architectural level separation where application and infrastructure level capabilities are provided as independent components with rich interfaces that completely encapsulate the underlying complexity and provide some degree of loose coupling and therefore pluggability.

However it would be a major mistake to assume this is purely a technical exercise to re-architect the IT environment into pluggable components which then, as if by magic, turns an inflexible business into an agile, adaptable environment. In this report we shall look at how better architecture, specifically Service Oriented Architecture, can improve an organization's inherent flexibility and response to change, together with the architectural decisions that need to be made by both business and IT architects to create the level of flexibility relevant to the needs of each enterprise.

SOA in Context

Real World Reuse

Underlying the extensive duplication, inconsistency and complexity there are in every enterprise many aspects of commonality. The duplication and inconsistency has occurred because we haven't had a system for managing commonality and the diversity in a structured manner.

Lets consider an example. In most enterprises there is a common core of customer and order data, processes and rules that need to be consistent. Consistency is important both to provide a uniform view to the customer as well as ensuring compliance with statutory or business policy requirements. But together with the need for consistency there are almost always requirements for variant business processes and data. All large enterprises are hugely complex with processes and data requirements that vary by geography, channel, product, market and so forth, and this variation needs to be managed such that the business organization is responsible for determining the mix of common and custom functionality.

In many enterprises there will be a business requirement for a small core of services to be made entirely standard for use throughout the enterprise. However the business processes using the services may vary considerably in data requirements and behaviors across different business units. In many enterprises there will be less clarity over the requirement for core business service standardization, and even in those that can decide on standardization today, they can always change their minds tomorrow!

So whilst reusing standard business services sounds like a good idea, there is a need to implement the standard services in a manner that allows continuous variation of the business solution and processes, that does not compromise the standardized parts.

Services and Component Architecture

Many industries including semi-conductor, electronics and automotive have adopted a component based approach in pursuit of standardization, cost reduction and time to market business strategies. We may observe that componentization and standardization occur in a more mature industry sector. At one time automobiles were entirely custom built. Today an automobile product line is comprised of tens of thousands of components many of which are traded and used in common with other product lines and manufacturers.

A component is a capability that offers a service via a published interface that provides a number of important benefits:

- The component has clearly defined dependencies and can be upgraded or replaced more easily because the impact of change is clearly understood
- The user view of the component is restricted to the external behaviors specified in the interface. With full encapsulation the component implementation can be changed without impacting on the component user. The component can also be replaced by the provider with alternative component(s) that may be cheaper or better performing.
- Full encapsulation also allows the user of the component to exercise choice – to use alternative components which may provide specialized functionality or different performance characteristics.
- The component can be assembled in a hierarchy of components that includes standard and specialized capabilities.

The service concept is based on encapsulated units of capability that offer a service through a formal interface. Web Service protocols provide a rich, technology independent interface structure that govern the use of the service including the description, security and transactional behavior. The service concept is therefore very useful in addressing integration problems, enabling a higher level of loose coupling than achieved using proprietary EAI. However the **service** should also be seen as an integral part of a component model – in which capabilities are managed as components that offer services. The SOA therefore enables change at many different levels, not just the coarse grained business service.

A service oriented application portfolio is analogous to an automobile or high tech product. Components are assembled together to form useful units of work using agreed interfaces which allow components from various sources and technologies to collaborate. Alternative components may be assembled together to deliver custom requirements. Consideration of the scope and dependency of a component is an important factor in service design as it determines the potential impact of change and therefore cost and response time.

Figure 1 – The Componentized Business

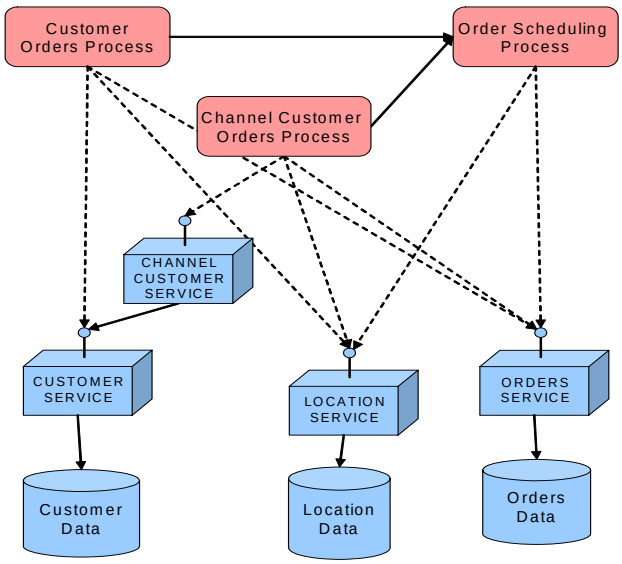


Figure 1 shows an example where standardized services are available for Customers, Location and Orders. The services offer operations on customers, locations and orders and are used by Customer Orders and Orders Scheduling Processes, which provide standardized business functionality for particular types of orders and customers. However in this example channel customers have specific contractual requirements which cannot be easily met by a common service and the variations are incorporated in a specialized Channel Customer Service.

The combination of the regular Customer Service and the Channel Customer Service might be considered an assembly which is

orchestrated dynamically depending on the context of the customer in the process instance. Note this example is based on business services, but a similar SOA may be used for technical infrastructure capabilities which provide similar assembly and reconfiguration benefits for middleware, network, storage, and other support services.

The SOA approach also allows a high level of flexibility at both design time and run time. Services may be upgraded or replaced more easily particularly if the change can be effected without change to the interface. Assemblies of services may be created dynamically dependent upon context which makes maximum use of standardized services and introduces specialization (flexibility) with minimum necessary change.

The service architecture is predicated on very high levels of reuse of pre-existing services and components. Ideally these will be pre-certified such that specific solutions can be developed very rapidly and implemented with a reduced horizon of change. Whilst the reality of an SOA is rather far removed from the pluggable concept of science fiction, it is a big improvement on the practices currently adopted by the typical enterprise, allowing continuous change of smaller units of functionality in projects that have radically lower time and resource profiles.

Understanding Requirements for Business Flexibility

A Flexibility Framework

The demand for flexibility is undisputed - however uncontrolled change is highly undesirable. The result of unconstrained flexibility is duplicate and inconsistent functionality and today's typical enterprise is a perfect illustration that flexibility costs. IT budgets everywhere are massively inflated because of unconstrained duplication of functionality and support costs, as well as the cost of management and integration of the disparate systems.

But total standardization will not work either. What 's needed is a blend of standardized and custom capabilities that allows standard services to be used and specialized by custom services to meet particular business unit needs. For this reason a framework is required that manages standardization and diversity, enabling flexibility where it is required and actively limiting flexibility where it is not needed.

Standardization

Standardization is a function of industry and organizational maturity. We can observe how standardization of railway gauges, clothing sizes and building industry guidelines, to mention just a few examples, has caused transformations of entire industries. SOA is based on the principle of standardization of services that can be used (reused) in many different ways across an enterprise or its ecosystem of partners, customers and suppliers.

In most enterprises today it is hard to determine standard services because there are so many alternative versions of the truth. Each business division, geography or segment will almost certainly see their business requirements as unique, and require the right to change as the business develops. On the other hand there are corporate level requirements to maintain a comprehensive and consistent set of services for core resources such as customers, suppliers, products, finances and assets that span individual business units.

Standardization in service architecture is occurring on many levels:

- The SOA approach is being made possible by the development of standard interface protocols that enable business and technology service interaction and management independent of the underlying platform technology.
- Many enterprises, recognizing the opportunity to share services are well advanced in developing standard definitions of data based on XML schemas.
- Enterprises will progressively implement standard services for core business functions that bring economy of scale (reuse) and consistency to diverse business processes.
- Ecosystems such as supply chains or partnerships, often encouraged by dominant parties, agree standard services that will provide a collaborative commercial network.
- Independent Software Vendors (ISVs) provide application functionality as service based components which enable greater choice of supply. In some cases a particularly dominant ISV may establish de facto standard services, which will encourage other ISVs to develop collaborative and or alternative services.
- Utility providers offer standards based infrastructure services.

Diversity and Differentiation

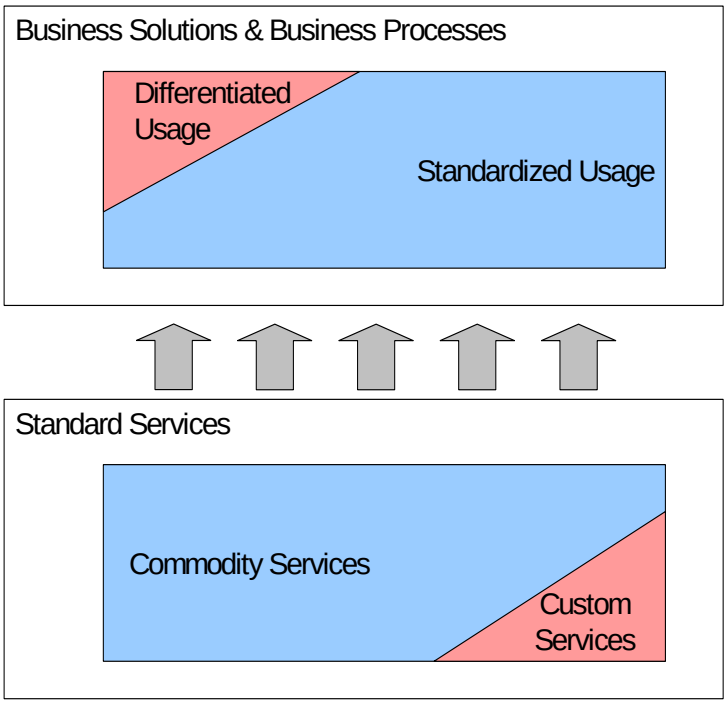
The challenge is actually very similar to that of the automotive manufacturers who want to establish the highest level of standard components, in order to reduce cost and time to market, while creating highly differentiated end products that meet their particular market requirements. However while automotive products are relatively stable, information solutions are subject to constant adaptation – adding new rules, types of events, attributes and so forth. Both of these requirements for change and variation need to be managed within a common framework in order to maintain the integrity of the standardized architecture.

ADAPTATION	CUSTOMIZATION
Change to standard services Enabling continuous change Responding to new requirements Response time critical	Specializing standard services Enabling custom solutions and variants Creating minimum increment to standard services Requirement critical
Maintaining architectural integrity	

Table 1 – Adaptation vs Customization

Today it is customary for enterprises to acquire enterprise applications for the majority of their business applications needs, and in many situations business processes are changed to facilitate the acquired product. These same vendors are progressively transitioning their application products to services and in the longer term it seems likely that all enterprises will use a high proportion of commodity services – functional capabilities that are largely undifferentiated from competition. However SOA introduces the potential to customize services whatever their

origin both by exerting choice at the service level, which is a much narrower scope than the typical enterprise application, as well as customizing and specializing the acquired services. The real area of opportunity therefore is how an organization takes advantage of the commodity services and focuses the use of custom built, and hence more expensive, services on the areas that matter.



Many companies use the idea of core and non core business areas as a basis for making strategic decisions, particularly outsourcing. Similar business domain classification systems may be usefully used to determine how services are both sourced and deployed. Figure 2 illustrates a model for thinking about these issues showing how the decisions around the provisioning of standard services is a quite separate issue from the use of the standardized services in solutions.

Figure 2 – Standardization and Differentiation Model

Policy in each of these areas is clearly a matter for business involvement. If an area of the business, a domain, business resource or service is considered non-core and a policy is adopted to utilize commodity services, there is a further decision on whether the **usage** will also be standardized, or whether differentiated usage will be permitted.

Requirements for Change

In the past there has typically been insufficient attention given to thinking about the future systems or process requirements. In an SOA environment it makes sense to consider what change is likely and to develop strategies that allow a well thought out response.

In many situations such as major reorganizations or mergers and acquisitions, IT is seen as an inhibitor, and the IT organization may be consulted only in terms of implementation. Consider how differently a business might be able to respond to potential opportunities if the IT environment has been engineered specifically for particular types of change, or if there is a considered architectural response to a wide range of scenarios. Table 2 illustrates a sample of change scenarios. Note this is purely illustrative, and is not intended to be comprehensive.

Requirement for Change	SOA Response
Merger & Acquisition – widespread duplication and conflict of functionality Requirement for standardization but commonly also extension of existing capability	**Classic response:** Select best of breed applications and standardize where possible. For other applications use integration approach **SOA response:** Tactical action - integrate core functionality using common XML schemas to normalize and exchange information between disparate applications. Strategic integration – adoption of standard components and services with customized services and processes to meet specialized requirements
New Product Introduction – where product characteristics considerably change the research, development, launch and support processes involving new data types and attributes, rules, events and value chains()	**Classic Response:** Creation of product specific applications; major modification of existing applications and or customization of enterprise applications. Possible new versions of existing enterprise applications **SOA Response:** Potential new standard business services, that complement existing core services. Create services that specialize existing and new standard core services.
New Channels – requiring significant variant business processes including events, rules, and data types	**Classic Response:** Create channel specific applications. Modify core applications to accommodate **SOA Response:** Create channel specific services that specialize standard services and processes
Real Time Enterprise (RTE)	**SOA Response:** Establish 360° service(s) for key resources. Migrate existing applications to use the new 360 ° services

Table 2 – Requirements for Change

Architecture for Change

Structure and Policy for the Flexible Business

The word service is a very overloaded term and open to misinterpretation because it is in such common use. Some enterprises even prefer to use the term **common capabilities** because it avoids confusion. Perhaps a more practical approach is to establish a structure and nomenclature of services that identifies the various classes of service more precisely such as illustrated in Table 3.

This example structure also includes a layered approach, which is a basis for defining generic policies for classes of services, and a basis for creating a systematic approach to assembly of services.

Layer	Main Role	Description	Operations
Process Services	Orchestrate other services; apply process- specific rules	Business process service that consumes and orchestrates one or more services from all layers. Examples: Orders Process Service; Channel Orders Process Service	User Interface independent, but designed for a specific business process
Core Business Services	Apply the enterprise's business rules	Business level services that offer a number of business operations. Examples: Customer Service; Assets Service; Orders Service	User Interface and business process-independent, so can be used in various contexts
Underlying Services	Apply the business rules, but not exposed as well-formed service	Used for services that will require a façade. Example: Enterprise application services for core business areas that will require customization. Example: Implementation specific services.	Highly generic or implementation-based, so its interface is not ideal for exposing to consumers
Utility Services	Shared by Core Business Services	Services that are intended to be widely used. Often commodity services. Examples: Calculations, Algorithms, Directories; ubiquitous underlying services	User Interface, business process and often domain independent
Infrastructure Services	Provide generic infrastructure services	Implemented as part of ESB architecture. Examples: Events; Security; Identity; Persistence	Highly generic and implementation based for widest applicability

Table 3 – Service Classification and Layering

Some example policies may include:

- **Service dependency** – managing dependency is a very important part of the SOA. The reduction in dependency is a vital element of creating a flexible application environment. The layered service architecture allows us to define rules for dependency first at a generic level – for example how services may interact within and across layers, then for individual services. Generalized rules will probably include a prohibition on cyclic dependency – i.e dependencies between A and C, and C and A, a pattern that makes testing really difficult. It is also recommended that policies are set that cluster services to create separation between groups of services, perhaps using domains or a similar modeling concept.
- **Service clustering** - a candidate policy using an appropriate clustering technique, for example domains, to prohibit inter-domain reuse in order to increase separation of

concerns. Inter cluster reuse to be effected via the process layer, in order to reduce the impact of change.

- **Standardization** – at the lower layers services will be more generic and standardized. Conversely in the higher layers services will become progressively more specific. Utility services will probably be made available on the widest possible basis, perhaps acquired as commodity services. Policies relating to classes of services should be confirmed for individual services in the service specification – classifying the specific service as core, commodity, and identifying the classes of services that can use the service.

Layered architecture is initially developed at a logical level interpreting business requirements as services, and establishing policies that directly reflect business requirements. The business perspective may then be mapped to implementation and deployment views creating traceability between the business service and the implemented service view. This traceability is vital because policies that directly reflect business needs for flexibility will be traceable to deployed services.

Services identification and design

The discussion so far has focused on preparing for change and specialization, because we know that change and variation is inevitable. However that shouldn't stop us considering how we can identify and implement services that are as stable as we can possible make them.

In IT generally there is almost unquestioning acceptance that requirements are derived from using the Use Case technique. Whilst the Use Case is without doubt a widely used way to understand a particular process or process step, it is not a good way to identify core business services because it is situation specific - the only generalization that can be determined is within the particular Use Case. Naturally we want core business services to be applicable to many different use cases.

Sean McGrath [1] suggests that the real trick is to get purpose-agnostic data representations of business concepts like person, invoice, bill of lading etc., flowing around processing nodes. This sounds like a worthy ambition but it needs some methodological support. One tried and tested way to achieve this is to identify core business services based on resources. For example customers, suppliers, products, assets and orders, which are in most companies foundational concepts and highly stable. Resources is a synonym for Business Types or Entities, which store information about the core business resources.

Typically resource based core business services can be highly generalized and designed for extension, for example using patterns such as Key Value Pairs [2]. There is also increasing interest in implementing 360^0 services that provide an enterprise wide view of these core resources with highly generalized operations that can be specialized by situation specific core business or process services.

Once core business services have been identified, Use Cases are a useful technique to identify process services.

Patterns for Change

> Coupling is a measure of the interdependence of modules. The higher the coupling, the more likely it is that changes to the inside of one module will affect the proper functioning of another module. . . There is no way to make modules in a structure absolutely independent of one another, but it is possible to come up with a structure that has so little coupling that you can *usually* modify one module without disrupting others. That is a structure that is *usually* easy to maintain.
>
> [2] *Tom DeMarco Yourdon Inc, 1978*

Service architecture addresses change by introducing loose coupling. But contrary to popular understanding a well designed SOA should create loose coupling throughout the entire application and infrastructure space, not just at a façade layer hiding monolithic and hard to maintain systems underneath.

In addition to structural and standardization policies discussed above, it is also important to consider policy in relation to potential customization and change, for which requirements may or may not be known. There is a wide range of actions that can be made in response to change requests. That is precisely the problem that IT organizations have faced in the past, as the specified architecture is compromised usually even before the application is implemented, as unforeseen requirements are implemented in whatever way is expedient.

As an invocable capability, a particularly useful characteristic of the service is that certain types of behavior can be manipulated at runtime, providing suitable controls are in place to ensure authority and compliance with policy. Some of these change patterns can be introduced irrespective of the service design. Other require preplanning at design time.

There are equally many ways that behaviors can be manipulated at design time, and it is relevant to consider that if the service architecture has been well thought through, the horizon of change will be much smaller than current convention, and therefore design time changes may also be effected very rapidly.

Here an analysis of the potential requirements for change is really required. Without it the architect is shooting in the dark, applying his/her best judgment of how services may need to adapt to change. But with a reasoned assessment of the business direction, the service architecture can be developed in such a manner that specific change and customization patterns are identified and enabled in the design phase. Further the architectural intentions may be recorded as policy, both in a general manner that is applicable to the overall or segments of the architecture, and also as policy recorded against individual service specifications.

Table 4 summarizes some of the relevant change and customizing patterns that the architect may consider. This list is not intended to be exhaustive.

Structure and Policy for the Flexible Infrastructure

This paper is focused mostly on the business service layers because this is where business flexibility is primarily going to be enabled or not. However it is important to consider the impact of the infrastructure on flexibility. The concept of the ESB has been widely reported as being an essential component of the SOA environment, because this delivers a platform and technology independent switching mechanism. Further the ESB layer also provides some very useful behavioral manipulation capabilities as discussed in the Patterns for Change section above.

However, the SOA architect also needs to consider how the wider infrastructure services are able to respond to change. This is particularly relevant as services become real time invocable capabilities, which means that response time to change will potentially need to be radically improved over current best practice.

In the early stages of SOA the infrastructure consists primarily of an ESB, Web Service Server and a registry which are integrated with existing infrastructure. Moving forward we must anticipate that all infrastructure capabilities are implemented as services in a layered architecture in which each layer encapsulates lower layers, simplifying utilization and management – mirroring the business service architecture discussed above. Individual services form components which provide atomic capabilities that can be reused in many situations and create an inherently adaptable architecture. Provisioning is an assembly based process in which services are dynamically contracted to provide resources to specific assemblies.

Design time change	Run time change
Service layering and sharing. Variants built using specializing software at higher layers	Key Value Pairs - selected business types are enabled to allow business type instances to store any number of key value pairs (attribute type name and a value). Sometimes referred to as UML's Tag Definition/Tagged Value [3]
Generalizing framework. Points in a program design where extra code can easily be inserted.	Service switching - appropriate service is selected depending on context, rules or events.
Workflow engine - enables new work flows to be defined, often using diagrams	Generic Service - generalized access service providing single interface to CRUD functions defined using XML schema definition.
Generic update operation - update operations designed to enable any attribute of an instance to be updated if supplied in the inputs	Metadata driven code - File of values that are read at runtime to find applicable rule or value dependent upon context
Operation extension - operations of services are extended to perform more processing or data, while not impacting existing consumers.	Rules engine – service calls rules engine (service) for processing – generally IF . . . THEN . . . ELSE format
Alternative Implementations - multiple implementations of the service, conforming to the same specification, can be deployed at different endpoints. (also runtime)	Service generalization - services and their operations are designed to handle a wider variety of situations than is necessary for present purposes. Customization occurs within higher layers
Façade - underlying or utility services are simplified for the consumer by hiding those services' interfaces behind "wrapper" code.	Document driven - import and export (standard) XML documents rather than parameters. Operations can pick and choose the document fields they need to work on.
Differentiated service - a service offers different behaviors depending on the context of use. (also runtime)	

Table 4 – Sample Change and Customizing Patterns

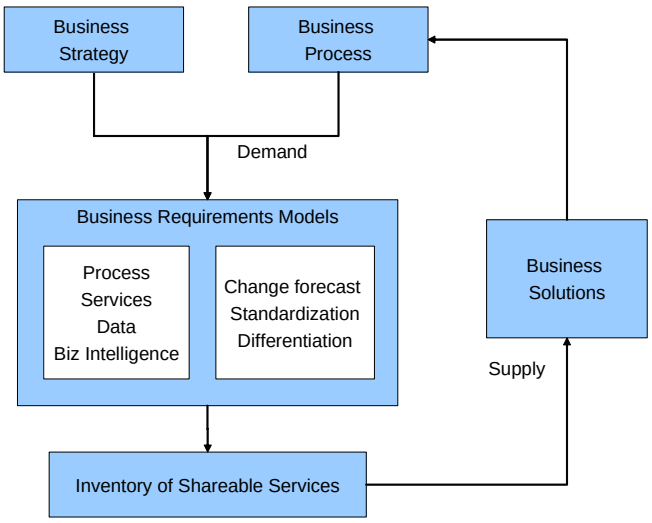
Similarly the infrastructure surrounding service delivery and provisioning is set to change radically. Consider the analogy of a supply chain ERP application. The ERP manages the execution of supply chain processes in a fully integrated manner such that all parts of the business are fully synchronized. Crucially the ERP depends on a set of managed databases that control the configuration and use of data and processes. We can anticipate that the future SOA will be managed by what we will refer to as an IRP – an Infrastructure Resource Planning application, which is a set of services (of course) managing the end to end configuration of the entire service life cycle, based on comprehensive metadata that records the state and status of all the moving parts. This will allow the infrastructure to adapt in the same timeframe as the business can adapt.

Organizing for Flexibility

Flexible business information services don't happen by accident. In addition to changes in architectural practice and techniques that can have significant impact on an enterprises capacity to respond to change, it's also necessary to consider organizational issues – changes in roles and responsibility.

Service Supply/Demand

Adoption of SOA requires significant change in the way information solutions are provisioned. High levels of reuse will only be achieved if investments are made to establish a significant inventory of common services and this requires radical modifications to project sponsorship, funding and organization. The investment approach together with the requirement for highly generalized services implies that at least some services will be delivered separately to business solution driven projects.



Planning for generalized services therefore needs a different form of interaction with the business, as illustrated in Figure 3, to understand not just the immediate requirements but also to establish an understanding of the requirement for flexibility. It's not sufficient to simply generalize the business type/class/service models, these technical level models need to be developed in context with known and candidate business strategies to obtain a clearer understanding of the real demand for service behavior – that can then be used to drive the architectural decisions discussed above.

Figure 3 – Enhanced Supply/Demand Model

Service Provisioning

Separation of supply and consumption of is a prerequisite for service delivery in order to ensure no dependency between specific projects and shareable services, sometimes referred to as the twin track organization. The new organizational model also requires clarification of roles and responsibilities around customization and assembly.

Metrics and Measurement

While flexibility is widely seen as a high ranking business requirement, the industry is only now starting to explore useful metrics. There are some general metrics that apply to SOA, such as solution response time, certification cost etc, but measuring flexibility and adaptability is less straightforward.

First, flexibility is only really going to be measured when business change is ultimately implemented, and it is likely to be extremely difficult to make comparisons between solution projects. And arguably retrospective measurement is too late. There's a cost to flexibility and what's needed is an understanding of relative flexibility as the architecture is developed in order to bring some realism to policy making decisions.

Table 5 provides some practical suggestions based on data that is available from models and the asset management database to support planning and solution delivery decisions that can improve the flexibility of the delivered services.

Flexibility Metrics	Rationale	Applicability
Numbers (and granularity) of services	Size of management task; Duplication of functionality; Difficulty of discovery	Architectural planning tasks Governance activity
Change impact	Maintainability – cost to customize or change	Architectural planning tasks Change project planning
Relative service independence	Architectural quality and cost assessments Cost for reuse, assembly and maintainability	Architectural planning tasks Design tasks Governance activity

Table 5 – Candidate Flexibility Metrics

Governance

Governance of the SOA is analogous to city planning – providing common services such as roads, power, water, waste disposal etc for widespread usage and policies such as planning guidelines and interfaces that allow the common services to be integrated into locally managed systems, buildings, homes, etc.

Like city planners, IT architects do not have total authority over their domain, rather they have to persuade their colleagues that a certain level of standardization is vital in order to avoid chaos, and to allow orderly and cost effective development of business solutions.

The parallels with city planning are very relevant because planning regulations have the force of law. Whilst some of these relating to aesthetics might be arguable, no one would dispute health and safety regulations. In the same way SOA policies should be seen as inviolate and compliance being absolutely essential. If the SOA policies are compromised, the overall flexibility of the enterprise may be jeopardized.

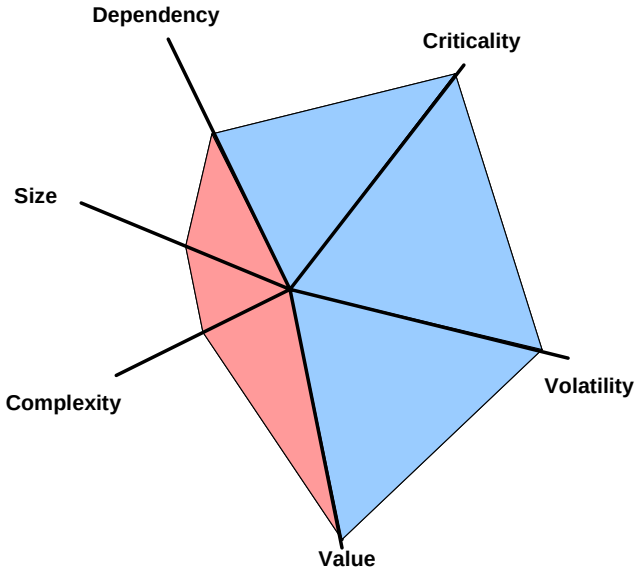
SOA policies as discussed above are unambiguous design decisions relating to dependency, service specification and usage and should therefore be implemented as attributes of the formal service specification. The policy attributes should be maintained in the asset database and registry and utilized to validate the use of services is compliant with policy.

Service usage will need to be validated at both design and run time. Checks on permitted usage ensure dependency and design pattern policies are complied with.

Enterprise Roadmap for Flexibility

Evolution Not Revolution

For most enterprises the transition to SOA will be progressive over a number of years. The technologies and standards underlying SOA are today reasonably mature and for most enterprises progress will be constrained by the ability and willingness to invest and establish standardized approaches in environments where standardization is not necessarily a business priority. And whilst business flexibility is widely seen as a high business priority by its very nature it is hard to justify and measure. Also in many enterprises there is a healthy level of skepticism for new IT concepts and technologies based on past experience and there is a need to demonstrate success.



So a realistic expectation in most enterprises is that there will be a staged process in which carefully selected services will be introduced to demonstrate the benefits, which will then enable in Early Stage Activity a more broadly based program to establish core business services and to rationalize priority areas. The question of prioritization therefore needs a balanced approach that minimizes risk while addressing volatile, critical and high value areas of the business that demonstrate the benefits, as shown in the spider diagram in Figure 4.

Figure 4 – Balancing SOA Risk and Return

Maturity Model

The implications of SOA are quite far reaching and it will be necessary for each organization to establish their own requirements and strategies. To manage this process of progressive change and adoption a maturity model is a useful technique in which desirable or planned states are identified at various stages of maturity as shown in Table 6.

The parallels with city planning are again highly relevant. In the same way that every city is quite unique, the type of flexibility and standardization required will vary greatly between enterprises. The distributed manufacturing corporation moving to establish a real time supply chain with its suppliers will have very different needs from a centralized financial enterprise focusing on business process improvement and consistency of customer information and services.

In the Early Learning stage it is likely that flexibility will be primarily enabled because of technical loose coupling of services and reuse of specific services that is made easier by the technology independence of Web services. Most enterprises will go through an Integration Stage during which they will be focused on exposing and publishing standardized core business services. At this stage the enterprise should be implementing flexible architecture policies and delivering cost reduction and business response improvements.

However, at the Integration Stage the underlying core business and legacy systems are likely to be largely unaltered, and the services exposed using façade and composite application strategies. While this will make service reuse easier, change of core business systems may be more difficult. It is not until the Reengineering Stage that most organizations will address this issue, and introduce loose coupling right across the application environment creating a genuinely flexible environment.

	Early Learning	Integration	Reengineering	Maturity
Management	Project based activity	IT investment in sharable services	Change in business practice	Business investment in shareable services
Architecture	Run time reuse	Run time and design time reuse and customization Layered architecture with flexibility based policy specified	Run time and design time reuse and customization	Automated governance over SO policy
Process	No change	Separation of provider and consumer organizations and processes Service Supply/Demand Flexibility Metrics	Service Engineering – service oriented life cycle	Infrastructure Requirements Planning (IRP) Automated change impact analysis
Infrastructure	No change	ESB and WS-STAR services Basic management and mediation	Transactional service behaviors Standards based service management Business service perspective from specification to deployment	Infrastructure as SOA
Flexibility	Platform independence Technical loose coupling	Widespread reuse of standardized services Simpler propagation of change Run time and design time customization and change	SOA at all application layers	Infrastructure services change in synch with business needs
Business communications:	Generally little change – tight focus on solutions	Status quo value chain and process design Requirements for standardization and competitive differentiation Requirements for flexibility Requirements for business process change	Service oriented value chain and process design Service oriented requirements Business intelligence and control improvement with real time information	

Table 6 – Maturity Model

Key Actions List

Top ten actions that an enterprise should take to start down the path to SOA based flexibility:

1. Define vision for flexible SOA and communicate
2. Establish business communication channel on the subjects of flexibility, standardization and differentiation.
3. Establish requirements for change
4. Develop Flexibility Maturity Model
5. Base the SOA roadmap on outcome of points 1, 2, 3 and 4
6. Develop an adaptation and customization methodology
7. Define layered architecture and service policies
8. Develop Service Portfolio Plan and apply policies
9. Implement governance program to ensure ongoing architectural integrity
10. Implement service metrics program

Summary

Introducing loose coupling between services is a definite improvement in application architecture. However this won't necessarily deliver the flexibility that an enterprise really needs. Service Oriented Architecture should be adopted with the mindset of a city planner, thinking about how future development and change can be accommodated by introducing sensible policies today.

Enterprises do not need unlimited flexibility. What they need is an understanding of how change may be implemented and at what cost. SOA provides the architect and business analyst with a language and framework to have sensible conversations about the economics of change, and how actions and costs incurred today may have a future impact.

Most organizations manage IT spend as a short term category. This has to change to allow a level of investment in shared, core business services. Equally significant is that the demand for shared services and the understanding of requirements for flexibility require a longer term perspective.

In the early stages the implemented SOA will almost certainly be skin deep. The superficiality is inevitable because restructuring core and legacy applications is not going to be simple or cheap. It will be made easier however by the façade layer, which will harmonize incompatible systems and allow deeper restructuring with less impact on the service user. But restructuring of the entire systems base is almost certainly going to be required at some time, and the priorities for this will be revealed by a deeper analysis of the real requirements for flexibility.

The level of planning involved in a flexible SOA may seem daunting to some. The level of analysis required to understand the real business requirements and to map these to models of the business in order to identify stable services and a framework for change may seem just too difficult. Is this too complex and too expensive a task to undertake? We suggest that the systems costs incurred by all major enterprises is almost certainly massively inflated by duplication and complexity caused by poor structure. The SOA, like a city plan, does not attempt to define every detail of every solution, rather it determines core features and crucially the connection points, such that the entire structure is capable of change.

Author

David Sprott is CEO and Principal Analyst of CBDI Forum Limited, an industry analysis and methodology company specializing in service and component architectures and practices.
www.cbdiforum.com

References

[1] Sean McGrath, Propylon

[2] Tom DeMarco, Structured Analysis and System Specification, March 1978, YOURDON Inc,

[3] UML's Tag Definition/Tagged Value - When it is necessary to introduce new notations or terminologies, UML provides user-defined extensions through the use of stereotypes, tagged values and constraints. Currently there are two extensions defined, namely Business and Objectory Process extensions. See Business Extensions in Business Modeling with UML, Magnus Penker and Hans-Erik Eriksson (ISBN 0-471-29551-5).