The Bloor Group

BEYOND SOA

How IT architecture can become more responsive to business needs

Robin Bloor, Ph D

WHITE PAPER

Executive Summary

That IT systems are inflexible is not up for debate. It is a fact, especially so in larger organizations that have multitudes of applications with a wide variety of functions. This paper discusses the nature of this problem, using the Service Oriented Architecture (SOA) as its departure point. Its findings can be summarized as follows:

- SOA and the computer standards that made up its foundation opened up the possibility of building better integrated software systems, in particular enabling software reuse and making it feasible to automate end-to-end business processes. However, SOA failed to address three specific problems that continue to be awkward sources of inflexibility within the enterprise. These problems are as follows:
  1. The incompatibility of data definitions and data model between applications and the difficulty in passing data between applications in a usable form.
  2. The lack of a mechanism for implementing global rules that need to apply throughout all applications or to whole groups of applications - particularly in the areas of application performance, business process performance and security.
  3. Despite the integration standards provided by SOA, application integration is still a problem with application connectivity often being provided on a point-to-point basis rather than being governed by an overriding architecture.
- In this white paper we suggest architectural enhancements to SOA which deals specifically with these issues. The architecture proposed here includes only a few extra components. In total it comprises:
  - An Enterprise Service Bus (ESB) capable of guaranteeing all program to program messages including the passing of data and distribution of IT policy as executable code.
  - An IT policy component for the central definition and distribution of IT policy.
  - A service monitoring component, which monitors both application performance and end-to-end business process performance.
  - A data integration component; to provide a library of data interfaces that applications can use.
  - A semantic data integration solution which enables the transformation of data in flight as it passes from application to application.
- We explain how such components would work and interact and we are convinced that the implementation of such an architecture would go a long way towards resolving the inflexibility of enterprise systems, ultimately enabling organizations to respond faster to business change.

The IT Conundrum

For many years, as the inventory of IT applications just mushroomed, the majority of IT departments built applications to address a specific requirement, with minimal connectivity and integration from one to another. From an IT perspective there was some sense in this. Siloed systems were easier to build and easier to manage.

But with the passage of time, and particularly with organizations wishing for good business reasons to automate, not just areas of activity, but end-to-end business processes, creating siloed single focus applications became less desirable and less viable. It soon became clear that if processes had to be better integrated at the business level they would have to be better integrated at applications level and at the IT infrastructure level. But integration was difficult to achieve and it generated complexity. Passing data or directives from one application to another or even having one application reuse the logic of another was fine if done only in a very limited way, but quickly became burdensome and difficult to manage if done extensively.

SOA: Towards An Agile Enterprise Architecture

The main attempt by the IT industry to eliminate application silos and yet address the complexity this produced was by means of the so-called the Service Oriented Architecture (SOA). This was a loosely defined software architecture that made it possible for applications to directly use each other's functionality directly. It was a clear improvement over what went before, and it provided a wealth of standards that enabled applications to connect to each other.

It introduced the idea of a registry (the SOA registry) within which individual business applications could declare (and define) their interfaces so that communications between applications were easier to organize. It gave life to the idea of an Enterprise Service Bus (ESB) - an important software component which could guarantee and becomes the back-bone of application-to-application communications.

SOA made it more feasible to design business processes rather than just business applications. But SOA also came up short in some important areas. It was at best, only half a solution. There are three specific areas where SOA proved deficient and which this paper focuses its attention on:

- **Data Models:** SOA did nothing to help smooth out the perennial problems created by incompatible data designs and data models.
- **IT Governance:** SOA had no architecture for implementing IT governance - i.e. The implementation of global rules and policy
- **Application Integration:** SOA provided no mechanism for managing the complexity wrought by integrating applications in a point-to-point manner.

Of Data and Data Models - And Change

Data is raw information. When we think of the data that business applications use, we think in terms of records; collections of items like customer name (or ID), customer order number,

order data, etc. We usually gather these records together in tables in databases. So databases embody a data model which is designed to suit the applications that the database serves.

Now, we might expect that all the data models embodied in all the business applications agree about the definitions of what a customer is, or what a product is, and so on. It seems logical that this would be the case, but it isn't the case at all. In fact such data models very rarely agree about this. There are many reasons for this, two of which are worth highlighting:

- **There is no "right" data model design for any collection of data.** There are best practices which many designers stick to that will result in the production of similar data models. But even if they agreed at that stage of their evolution, the model is often compromised to suit the capabilities and quirks of specific database products. And even if that were not so, any hope of standardize data models would be sabotaged by the fact that organizations use software packages for many business applications - and they have no control over the data models employed by such packages.
- **Applications are contextual and so are data models.** A further disparity in data models arises from the fact that a data model is contextual. Businesses often have, for example, multiple customer databases with each having different definitions of Customer. What should the customer ID be? A web-based application may take the email address as the unique customer identifier, whereas as a call center application may regard the customer cell phone number as the primary identifier. And that's before each application considers what other information is important.

Together, these two facts of life hobble any possibility of arriving at a common data model by simply combining all of the data models from all the databases that the organization uses. There could be as many different data models as there are applications. But even if there were a good level of agreement at the outset, data models change with time. And it's not just the entity definitions within the model, but also, sometimes, the relationships between data entities. Changes occur for many reasons. The real world, which is what is reflected in any data model, changes and business processes change to conform to it. Companies implement new technologies and bring changes to data models. Everything is, to some degree, in flux.

This reality creates problems that need to be addressed outside of each database. In particular:

- Applications need to exchange data in such a way as to ensure that the business meaning of the data and its semantics are not lost in the exchange.
- Data in motion needs to be properly prepared for its receipt. It is not the same as static data since it is intended for a specific application context. It needs to be available at precisely the right time in a usable form.
- Changes to data definitions need to be managed with considerable care because they can ricochet through many operational systems. Changes must be closely managed.

The Policing of Policy

IT governance has become an important focus for most organizations. It is a natural impulse and good practice to want to formulate and implement general policies for how IT applications are deployed and used. There are two distinct areas in this:

2. **IT Service and Performance**: An example policy might be that the eRetail application must deliver transaction response times in under one second and be available 24/7.

3. **IT Security**: An example policy might be that all applications must have a valid user login before they can be used.

Declaring policy is easy enough, but implementing it usually isn't. The difficulty it presents is illustrated in Figure 1. Policy is decided centrally but has to be implemented locally, in each application to which it applies, or possibly across several applications at once. What is more, the implementation has to take account of the fact that the policy may change with time.

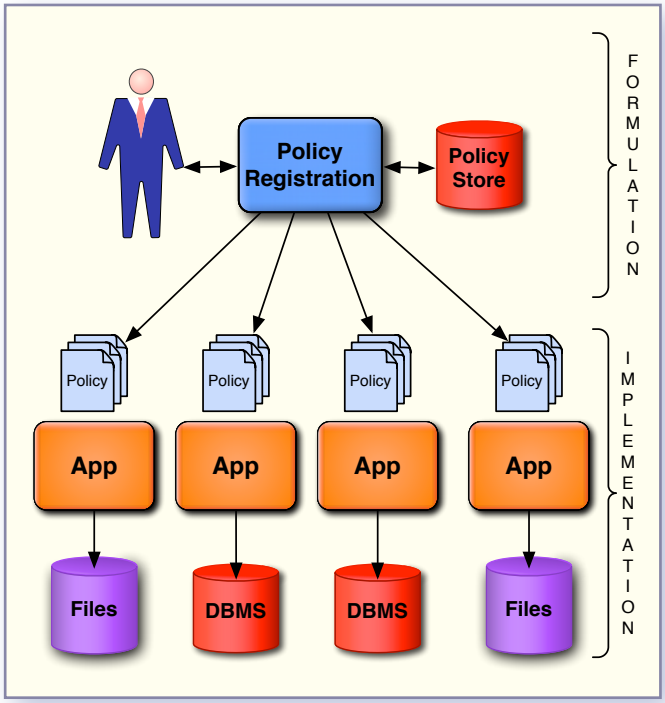

Figure 1: IT Policy Implementation

Consider the policy that: ***The eRetail application must deliver transaction response times in under one second and be available 24/7.***

Implementing this directive is not trivial at all. First we need to monitor the application to detect when performance starts to be sluggish and we must be able, automatically, to provision extra resources to the application should that occur. Next we need to know all the other applications this the eRetail application depends on and those which could (if they failed or slowed down) impair the performance of the eRetail application. We need to monitor their service levels so we can automatically respond if they deteriorate.

That's already complex, but now consider what will happen if we decide to include the eRetail application in an end-to-end business process for managing and fulfilling orders, which must also be available 24/7 and which mandates that order placement to fulfillment must be no worse than 4 hours. Now we have another service level for the eRetail application to satisfy, which has other dependencies.

If it had been possible to manage the service level of the eRetail application locally, it is no longer so. Monitoring the application and managing its performance has become difficult.

Now consider a specific IT security policy that: ***Only approved IT users, internal or external, should be able to see Social Security Numbers and Account Numbers.***

The situation on the ground could be far more complex than the simple policy statement ever dreamed of. First of all, as stated, it applies to every application that has access to such data, and not only call center applications and CRM applications, but any BI capabilities that have an ability to access production databases - or any other data stores that hold such information.

When you analyze the reality of such a policy statement, you may conclude that it's a demand to add program logic to many programs and to set specific security rules to several databases. That will take time. But worse than that, how will it be possible to keep the policy in force to

account for every change made to any of the systems affected? A policy statement can impose a requirement on every application the business runs, and that may necessitate a good deal of implementation and maintenance effort.

That is the fundamental difficulty with IT policy of any kind. Policy might be easy to define but it often needs to be implemented in many places and kept current in perpetuity. To exacerbate this problem, we face the reality that governments are increasingly imposing regulations on organizations, many of which have a direct impact on IT. This naturally ratchets up the number and diversity of IT policies that a company has to manage and control. In recent times this has brought us to the idea that all data should be governed in respect of its usage, accessibility and lifecycle.

IT policies are both a necessity and a headache.

Integration and Orchestration

It would be an awful lot simpler if applications didn't have to interact with each other. However, this is inevitable for two reasons. First we have the fact that one application may need to use the data of another application. This is a very common event.

Secondly we have the fact that if we formulate policy then we really will need a way to distribute the modules of software that enforce policy to specific applications in one way or another. Clearly, that can lead to a complex software environment.

But the situation is worse than that.

Businesses do not operate as a set of individual employees running various software applications, they operate by focusing on and refining their key business processes. When a customer buys something, the whole process:; placing the order, receiving the payment, dispatching the goods (or delivering the service that was purchased) and so on, may involve many activities, and hence it may involve many applications. This is a work flow, whether it has been formally designed and implemented as an IT work flow or whether it happens manually, with everyone in the process chain knowing what their role is and carrying it out.

It is a nice idea to imagine that when an organization is founded it can quickly implement all the business applications it requires and design all its work flows to suit them. Then, as it grew, it would only need to hire more staff and buy more computer power to cater for the increased workloads. But that never happens.

Even in green field situations, what occurs is a gradual increase in complexity combined with growing organizational entropy. Business processes change and new applications are introduced here and there to cater for the changes. Databases are changed and applications are updated. Integration requirements that were never contemplated when the organization was small, suddenly become necessary.

Usually the IT department responds by trying to fix each new problem as it arises. It makes connections between programs that were never previously connected so that a specific program routine can be called or so that specific data can be passed.

This is illustrated in Figure 3. (on the next page).

We can think of the applications as having been implemented in independent silos with no integration between them. App 1 may want to get at the data of App2 or perhaps a specific

program routine, so there are two possibilities. But notice that if App2 wants to get at either the data of App1 or some program routine in App1, it cannot necessarily be satisfied by the connections App1 made to App2 because the programing and data interfaces may not be the same.

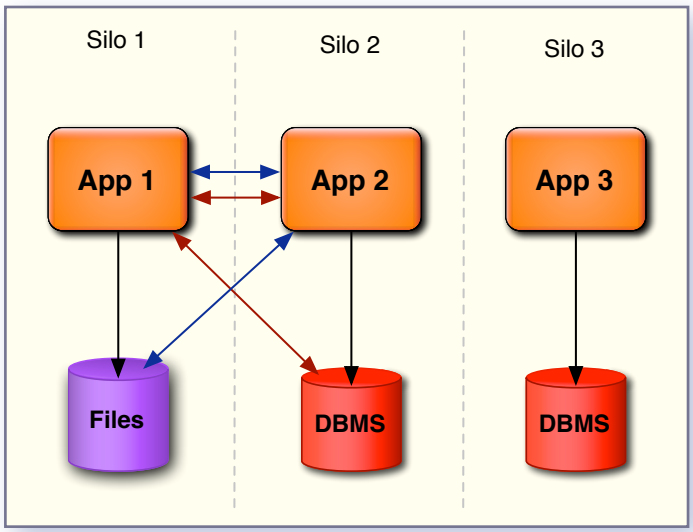

Figure 2: IT Integration

In practice, the building of such connections, necessitated by genuine business requirements, continues from month to month and from year to year until what once looked like a nice collection of largely independent business applications now looks like a bowl of spaghetti. Lots of applications are connected to other applications and it's all in a big tangle. The applications need to communicate, but sadly there is nothing available to organize their communication - no fundamental architecture which facilitates and manages their communications.

The Legacy Silo Architecture In Summary

We can summarize the situation in which most organizations find themselves in the following way. Most of the company's applications were written and deployed in silos. Insofar as there was any architecture, it was a silo architecture. Each application was built together with all the supporting software it needed for back-up, failover, data archiving and so forth, but that arrangement provided nothing to enable one application to connect to another.

The introduction of SOA helped, but its impact was limited. In particular there were three important problems that it never addressed. These are easy to describe, but not so easy to fix.

- Sharing data between applications is difficult because of incompatible data designs and data models.
- Implementing global rules and IT policy ***in an automated way*** is difficult because there is no global mechanism for sharing functionality that may be applied in many places.
- Integrating applications so that they can reuse each other's functionality is difficult because there is no common mechanism to enable such integration (despite the fact that SOA has provided some useful standards to assist in such integration).

The effort to solve this problems on a case-by-case basis, rather than through a common architecture leads inexorably to complications and escalating costs. The applications gradually become more complex by virtue of more and more ad hoc interfaces being built to connect one to another. IT policy is not implemented in a coherent fashion and data disparities occur from one application to another. And as a company grows and deploys more and more applications, the problem escalates, ultimately inhibiting the ability of business to respond quickly to change.

Of Architecture and Infrastructure

Frank Lloyd Wright once observed that "You can use an eraser on the drafting table or a sledge hammer on the construction site." The spirit of this warning applies equally well to IT architecture. Surf the web and you'll discover a plethora of examples of architectural errors that were only discovered when it was too late. Ask the right questions to any group of CIOs and you'll discover copious examples of inflexible systems with inadequate infrastructure to support them.

The Nature of Infrastructure

In IT, just as in construction, to a great extent, infrastructure is the realization of architecture. When you think of impressive architecture, it is natural to think of the aesthetic appeal of famous art museums or palaces, but for most buildings, that is not what the architect spends his time on.

A skyscraper, for example, is defined by its foundations and its supporting pillars, plus its entrances, stairwells, elevators and service rooms, plus its water systems, waste systems, electricity systems and communication systems. All of these are critical to the building. But exactly how the building is occupied can vary considerably. A single business might inhabit it, or it may be split into a plethora of small units.

When there are faults in the infrastructure they may not be so easy to correct. If the foundations of the building start to give, fixing it will be a trick and a half. If the waste systems and water systems are inadequate, or the air conditioning can't cool the space it needs to, then it is not going to be fixed with tape.

The parallel carries directly across to IT. In IT, the realization of architecture is also infrastructure. It is not too difficult to build applications. We're not making light of that, some applications may be big projects, but most of them are not. They can usually be built fairly quickly and will work well in a stand-alone manner. But if they are going to work with each other in an orchestrated manner, they need architecture.

So what can be done?

Towards A Responsive Business Architecture

In a sense the problem is simple. Applications need a mediation service that manages the requirements they have of other applications. In particular:

- They need a mediation service that is capable of connecting them to any other application to which they need to communicate.
- They need a mediation service that can transform data sent to them by another application so that it arrives in a form that can be used immediately, and which also transforms data they may pass to another application for its use.
- They need a mediation service that can invoke policies. Depending upon the policy this might execute the policy elsewhere and send back the result, or it might deliver a module of code that might be executed locally.

- Additionally they need the mediation service to guarantee every connection, every message passed, all data delivered and every service called. And the service needs to be intelligent enough to route all this interaction from one application to another swiftly and securely.

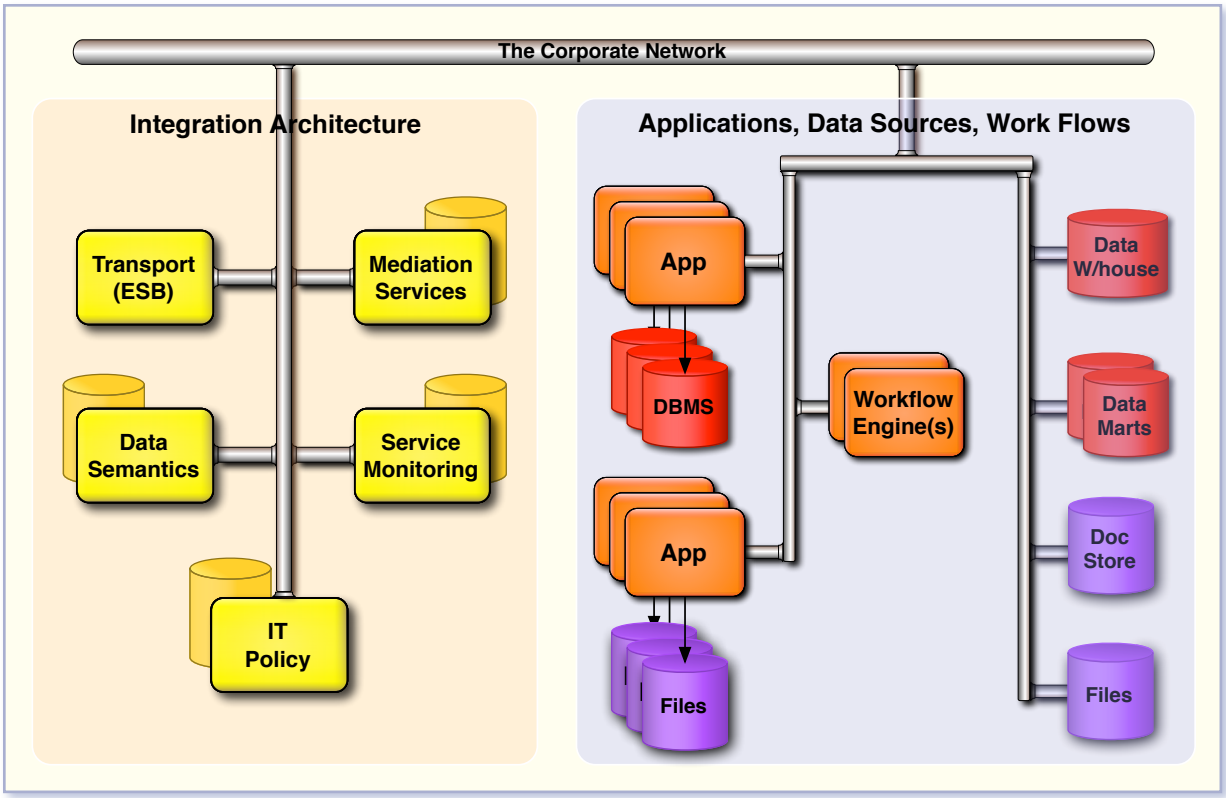

Figure 4: Application Integration Architecture

The applications need an ***integration architecture*** that embodies such a service and links together all participating applications. Without such an architecture, the only way to try to implement what we have described is by building individual integration adapters on a point-to-point basis. The coding of such adapters will inevitably be complex as it will need to manage app-to-app connection, message passing, data transfer and policy implementation for each application it connects to. In truth, this is what many organizations have done to some degree. Sadly this approach doesn't scale. At some point the complexity of writing and maintaining the code becomes overwhelming.

The integration architecture manages all this activity as a set of services that are available to every application, allowing each application simply to make calls to components of that architecture when it wants a specific service from it.

This is what we have illustrated in Figure 4. On the right hand side, under the heading **Applications, Data Sources, Work Flows** we depict all the corporate business applications, including the workflow engines that integrate business processes and typical data sources including a data warehouse, data marts, document stores and so on.

On the left hand side we have loosely depicted an integration infrastructure that serves the various applications and business processes. The first point to note about this architecture is that it embodies a transport management capability of the kind delivered by an Enterprise

Service Bus (ESB). It's role is to route all connections, messages and data to their intended destination. ESBs have other capabilities some of which may directly cover some of the other components we have shown in the diagram.

Nevertheless we will describe the other components as if they were separate. The most important from an architectural perspective are the mediation services. The specific services that an integration architecture needs to provide are:

- It must cater for all types of interaction; synchronous and asynchronous requests and responses, publish subscribe, store and forward, and batch transfers.
- It must manage sequence so that all messages, data transfers and connections happen in the right order.
- It must provide a comprehensive fault management service so that any failure of any element in any interaction is rectified. And this includes any failure of any component of the integration architecture itself, not just application failure.

The other three components shown are what their labels indicate. If data is going to be transformed "in flight" when passing from one application to another, there needs to be a component that has a repository of all data models and data interface requirement across all applications. Similarly there needs to be an IT policy component that can implement IT policy by furnishing links to processes that implement them, possibly even having the capability to distribute instances of such processes via the ESB. And finally there needs to be a service monitoring component that perpetually monitors service levels, not just for individual applications, but also the service levels of all end-to-end business processes. Such a component will invoke the fault management mediation services should any service level deteriorate.

Finally, it is worth noting that this integration architecture must be designed so that it is distributable across a network, even spanning multiple data centers and it needs to able to scale out as the traffic it manages increases.

The Responsive Business and Responsive IT

SOA wasn't so much a failure as a halfway house. It delivered benefits. It introduced standards that enabled better integration and it delivered a service based environment in which application capabilities could be reused to some degree. The integration architecture that we are advocating here is not a repudiation of SOA, it is an evolution of SOA.

Indeed some of the architectural components we have just described are already deployed within SOA environments. For example, many companies have deployed an ESB and in doing so may have also implemented some of the mediation services we have described - fault management in particular. Additionally some organizations will have deployed some service monitoring capabilities. In short, part of the integration architecture we have described here may already be in place.

But also many organizations have run into problems with their SOA implementations, and in most cases those problems resulted primarily from by the lack of a comprehensive mediation service. It became increasingly expensive, in time and resources, to integrate applications in the way that the business wanted and required.

Businesses need to be responsive. They live in an environment that changes rapidly and at times unexpectedly. They are subject to general economic pressures. Demand for their products or services may escalate rapidly and they need to seize the opportunity. Or the economy may turn down and they need to draw in their horns. Increasingly they are subject to regulation and they need to respond to the challenges it presents. Competitive pressures are greater than they have ever been. We live in an age of innovation and innovation breeds competition. And for many businesses competition is now international.

The force of innovation has conjured up more than new competitors. It has brought a whole new wave of social connectivity which is transforming the nature of customer relationships. And the pace of innovation in IT itself has not slowed. New applications, mobile computing, embedded systems - and more and more data to manage and analyze. Change touches everything and organizations need to respond.

The IT problems we described in the first part of this paper are problems of change. Applications need to change for business reasons. The motivation for such changes may be reactive, such as the need to satisfy regulatory requirements, or it may be proactive in pursuit of a business opportunity. But either way the business needs to react swiftly and effectively.

The integration architecture we described is intended specifically to accommodate change. It is an extensible architecture and it scales up and out. It is designed to be distributable across multiple domains. So while it might be implemented first to manage a small number of business applications, it can extend to support all applications within the data center, and if the organization has multiple data centers, it can span those. It can orchestrate business applications up and down the supply chain. It can reach into the cloud.

It is instrumented so that the performance of critical business applications is continually monitored and measured, and failures are quickly rectified. It accommodates two of the most difficult kinds of change that business systems need to accommodate: changes to data definitions and changes to IT policies. Ultimately it enables IT to respond quickly to change and that in turn enables the business to respond quickly to change.

As a final note, it is important to understand that what we have described in this paper is not a theoretic architecture. All the components described are available as commercial software products in one form or another and all the capabilities described can be delivered. It is possible to assemble such an architecture and most IT Departments would profit from doing so.