lustratus

Research

Message-driven SOA

Getting rapid results with SOA

Author: *Steve Craggs*

July 2008

Version 1.00

Table of Contents

Executive Summary

It sometimes seems like every company in the world, of any size or industry, must have heard of service-oriented architecture (SOA). It constantly appears in IT press, analyst and vendor marketing materials, every trade show appears to make at least a passing reference to it, and there is a steady stream of firms prepared to stand up in public to declare their own successes with it. What is more, unlike many IT initiatives, SOA even crosses the boundary between IT and business disciplines, with business executives attracted to the improved IT alignment with business objectives that it offers, from the ability to manipulate IT-basked business processes directly to the improved visibility of business performance delivered by the operational IT systems.

However, not all companies are in a position to take on the considerable challenges of full-scale SOA-based transformation. Many companies find themselves held back by funding and resource restrictions combined with a lack of knowledge and maturity in SOA activities such as process modelling and re-engineering. These companies tend to be locked into a highly pragmatic mode of operations, with every dollar of investment linked to a tactically driven return. But the value of SOA need not be lost to these companies. A new approach to delivering pragmatic, SOA-based value has evolved, combining some of the best features of SOA with event-driven principles – message-driven SOA.

Message-driven SOA has its roots in the ongoing requirement many companies have to improve integration across different business disciplines and systems, but by bringing in aspects of SOA it offers an easier and more cost-effective response to this need while providing additional SOA benefits such as faster time to market and improved business visibility of operations. At the same time, message-driven SOA offers a tactical response to immediate needs that also remains consistent with the longer-term strategic goals of full SOA adoption.

Message-driven SOA aims to provide a subset of SOA functionality that is simple and yet effective. It may not be for everyone, but it certainly provides an important option for companies struggling to get onto the SOA path.

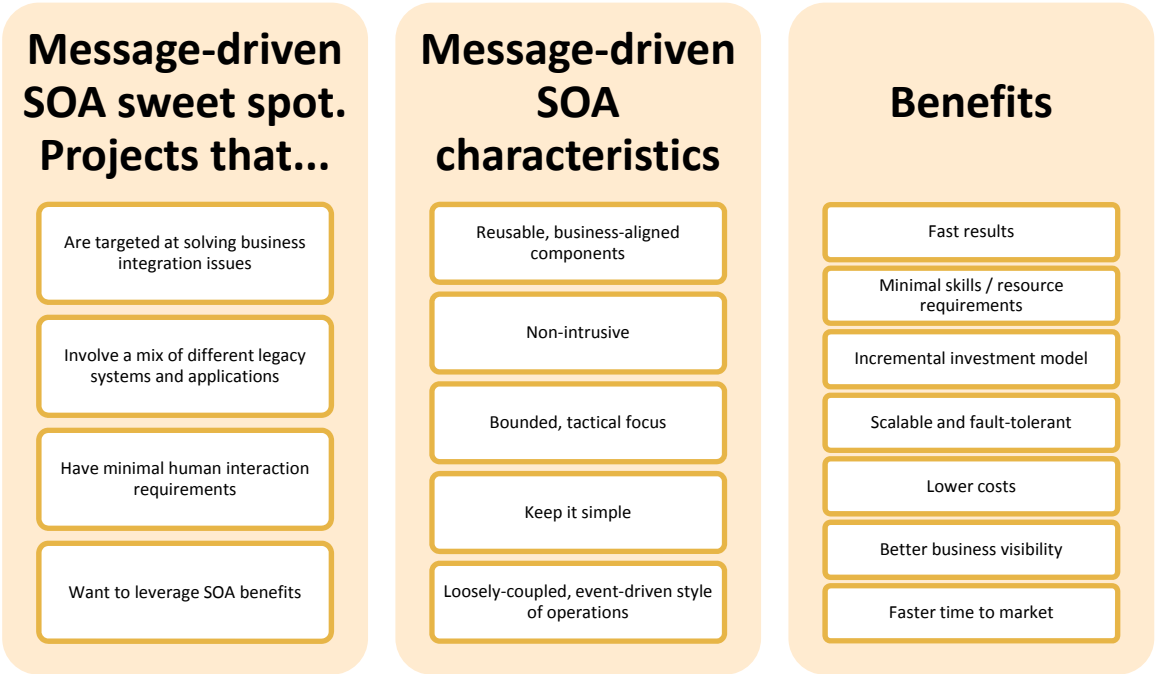

Figure 1: Message-driven SOA summary

SOA Today

Service-oriented architecture (SOA) is great. Everyone loves the idea, it seems, and countless companies across the world are starting to work with it. Just to recap, SOA is all about taking pieces of code, programs and data references and packaging them up as reusable 'business services' that represent the execution of some particular business function. The SOA infrastructure ensures that these services can be called from anywhere through a standard method of invocation, and what is more the inputs, outputs and functionality of these services are crisply defined, turning them into 'black box' components that can be shared across a range of business operations.

The idealized picture of a full SOA implementation shows IT-enabled business processes broken down into individual process steps, each represented by a reusable SOA service, with meta-data controlling the flow of the process across the various services. Redundancies are removed and reuse shoots up, bringing down costs and speeding project delivery. The alignment of SOA services with specific business functions makes it possible for users to view operational activity in business terms rather than technical ones, enabling them to generate substantial value in areas such as compliance and management, risk mitigation and process efficiency. Related technologies such as Business Process Management (BPM) can optionally provide graphical, intuitive business flowcharts of the desired process execution, allowing processes to be modelled, analysed and changed by business-oriented people such as business analysts. Business Activity Monitoring (BAM) can provide dashboards reflecting business operations status against key performance indicators, and combined with business event technology can drive automated responses to pre-specified business conditions or alert the relevant personnel to take appropriate action. Business Rules Engines (BREs) can provide clearly understandable rulesets that will govern operational activity, implementing corporate policies as desired.

Benefits promised by SOA therefore include agility, flexibility, responsiveness, better alignment of IT and business goals and a clearer business-oriented visibility of IT-based operations. But as more and more companies start to get involved in SOA, some users are having considerable problems. Sadly, it seems that a growing number of companies have found themselves unable to implement SOA successfully, and are starting to get extremely frustrated by the experience. In essence, the most common feedback from these companies is that SOA is 'just too big and too hard'.

SOA challenges

Early experiences with SOA have been positive, with a number of major enterprises coming out publicly to detail very impressive results. This initial success has continued, with hundreds of companies successfully adopting SOA, at least for key parts of their businesses. But as SOA adoption starts to spread from the more visionary companies, who are prepared to commit serious levels of resource and time to achieve their goals, to the more pragmatically minded majority, a growing number are standing back in confusion. For these companies, investments need to be rigorously justified, with minimal risk. An incremental model of deployment is preferred, where investment can be staged while at the same time benefits at each stage can be validated. Ease of use becomes another key factor, since these types of companies do not generally have access to the same volume of skills and resources that the early adopters enjoy.

There are primarily five main challenges that tend to stop the more pragmatic companies in their tracks with their SOA plans:

- They lack maturity in the wide array of SOA concepts and tools
- They view SOA as requiring significant investment before benefits start to flow
- SOA plans get tangled up with process re-engineering and business transformation
- SOA architectural, design and operational skills are scarce and expensive
- Getting the existing software infrastructure to match SOA needs can be a big headache

However, a new SOA initiative is underway that is gaining increasing momentum amongst this group of companies. This new initiative is s simple, quick, way of approaching SOA that offers at least some of the SOA benefits to a wider customer base – message-driven SOA.

Introducing Message-Driven SOA

It may seem rather trite to talk about 'message-driven SOA' – after all, the Enterprise Service Bus (ESB) concept at the heart of most SOAs is built around the idea of a message-based communications pipe. It is this component that is responsible for linking different SOA services together, for example. However, the term 'Message-Driven SOA' is used in this situation to describe a different way of approaching SOA, contrasting with the process-based approach common today in most SOA marketing literature.

Process-driven SOA

In just about any presentation or paper on SOA, it wont be long before a screenshot of a business process flowchart or something similar turns up, showing how IT implementations of business processes can be changed at will by dragging and dropping process steps into the flow. Many of the highlighted benefits relate to the agility and flexibility offered by being able to manipulate processes in this way, and when coupled with human-oriented workflow and business monitoring tools the goal of streamlined, automated, efficient and continuously improved processes looks a real possibility. These process-oriented messages extend to wholesale business transformation, with the point being that the flexibility and power of this type of control over IT and human-based implementation of process changes enables companies to radically change the way they operate, although this has to be coupled with changes in working practices too.

These are all valid arguments in building the case for SOA – indeed, the technology is an immensely strong enabler of innovation and transformational change. It is also the case that some of the biggest success stories with SOA have involved companies who have fully embraced this process-driven view of SOA.

However, many of the more pragmatically minded companies find this vision all rather terrifying. The whole area of working out what services to build, coordinating with other business departments to define the desired functionality for the shared service, understanding the process flows of existing IT-based applications, finding skilled personnel that can be trusted to manipulate processes safely and of course comprehending the scary area of transformational change is just too much. These companies may look enviously at those more entrepreneurial firms who are seizing SOA with both hands, wishing they had the same access to resources, funding and executive commitment. But does this mean SOA is not an option for them?

Message-driven SOA

Fortunately, SOA still has much to offer, even for companies not able to immediately strike out for full-scale business process Nirvana. A new approach to SOA has emerged, driven by a pragmatic need to improve business integration while also getting some of the SOA benefits, without having to tackle a lot of the bigger SOA issues. This new approach is usually referred to as message-driven SOA.

Starting with the basics, two of the most fundamental attractions of SOA, are as follows:

- The idea of reusable services, accessed in a standard way, is a good thing since it reduces redundancy and hence maintenance costs, cuts back on development costs for new projects and improves the quality of service IT delivers to the business
- Relating services to a particular piece of business functionality offers opportunities to get a clearer picture of what is happening in production operations at a business level

These aspects of SOA are highly desirable to many companies. However, the pragmatic point of view demands more. To make SOA more accessible to a wider range of adopters, the following characteristics are critical:

- Implementation should be as non-intrusive as possible
- Focus should be on delivering quick returns with less investment
- Responding to well-defined but bounded business needs is preferable to adopting major strategic and transformational shifts
- Every effort should be made to minimize complexity, even at the expense of more function
- Resource and skills needs should be minimized where possible

To be fair, there is one further requirement worth mentioning that harks back to the full SOA story. Although the pragmatic investor may be looking for a quick, easy, 'good enough' type of approach, there is still a strong need to be comfortable that the greater strategic goals are achievable. That is, this pragmatic use of SOA should not lead the adopter into a dead-end strategically, but should be a step in the right overall direction.

Message-driven SOA is a practical use of SOA that fits well with these objectives. Everything is driven from a clearly defined business need – the more bounded the better. A natural fit for this pragmatic need, bearing in mind the features that make up SOA, turns out to be some form of improved business integration. The old enterprise application integration (EAI) market was an attempt to respond to this critical need, but a lot of companies struggled with EAI complexity and cost. However, bringing in some of the SOA advances such as standards-based, peer-to-peer messaging and reusable business services makes for a more powerful and accessible solution. So, in message-driven SOA, SOA services are created as required, communicating with each other through the SOA message bus (usually an ESB). In essence, when a component wants to link with another, it does so by sending a message. Often, the message will be asynchronous, triggering some connected activity elsewhere in the system, in the manner of an event-driven architecture (EDA). Messages may be driven by the application, or in response to pre-defined business events, such as an inventory running low.

Contrasting process-driven and message-driven SOA

At a high level, process-driven and message-driven SOA approaches reflect differences in emphasis and purpose. As discussed earlier, process-driven SOA is often driven by a desire to deliver a more flexible and responsive IT infrastructure that is more efficient and cost-effective, aligns better with business goals and offers a higher degree of business agility. In other words, a strategic decision is made to adopt an SOA philosophy, and pressure is then put on individual projects to be implemented in SOA terms. The target is SOA adoption, in order to gain all the benefits SOA brings.

Message-driven SOA is not really about a company deciding that SOA adoption is the goal and then driving the implementation through successive projects, but instead is a tactical response to an immediate project need that also takes the opportunity to leverage the SOA model.

The difference between these two perspectives may seem subtle to some, but it is essential in understanding the values of process-driven and message-driven SOA. It also clarifies the approach to determining what functionality is required to deliver a message-driven SOA solution. Whereas process-driven SOA defines its functional requirements working downwards from the needs of a comprehensive, enterprise-wide deployment, message-driven SOA requirements build up from the need to solve specific integration-related problems while keeping to the general SOA theme.

At a more detailed level, there is a fundamental difference in the ways the two models operate, which has far-reaching consequences. With process-driven SOA, the desired component flows required to execute particular business services or processes are specified in some sort of process execution language, for example BPEL, and stored in a repository. When the process is called, a process execution engine calls up the required flow

and drives the operation accordingly. However, each instance of this activity is carried out in a single thread. Typically in an SOA environment, each separate flow will be executed synchronously within its own thread.

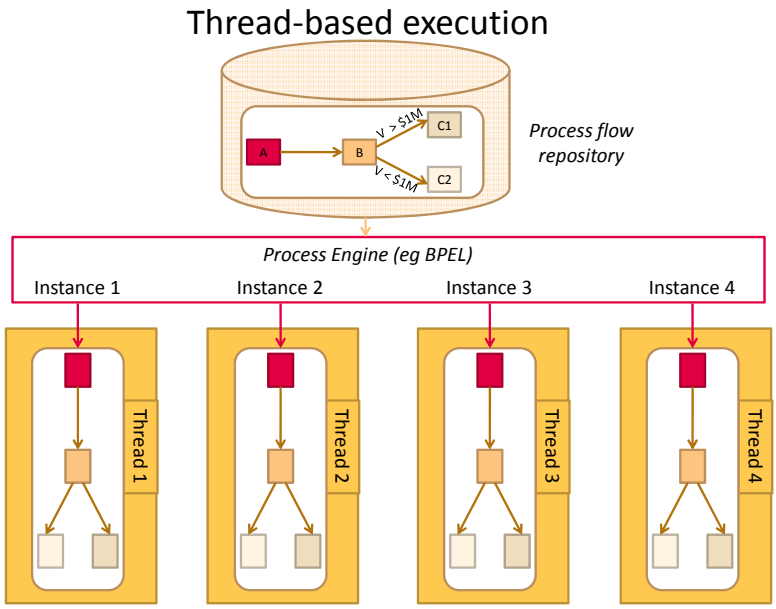

Figure 2: Process-based SOA executes one copy of the defined flow per thread

With a message-driven approach, however, components are linked by asynchronous messages being placed into queues. Typically, logical flow requirements are built into the messages, although not in the actual content but in the surrounding envelope. In this way, each stage of the process is decoupled. As a result, the previous restriction of one thread per execution of the process is removed, allowing multi-threading flexibility across all the flow components. Clearly, this has implications on such areas as scalability and performance. For example, if one component of the flow is running slowly, the message-driven approach offers the opportunity to start other threads, either on the same hardware or other platforms, in order to drive additional throughput. Since each component of the flow is simply reading messages from the input queue and processing them, there is no issue with having multiple instances in multiple threads provided there are no serialization requirements. In fact, this also allows additional hardware to be deployed to resolve a loading problem, as long as the new hardware can access the input and output queues.

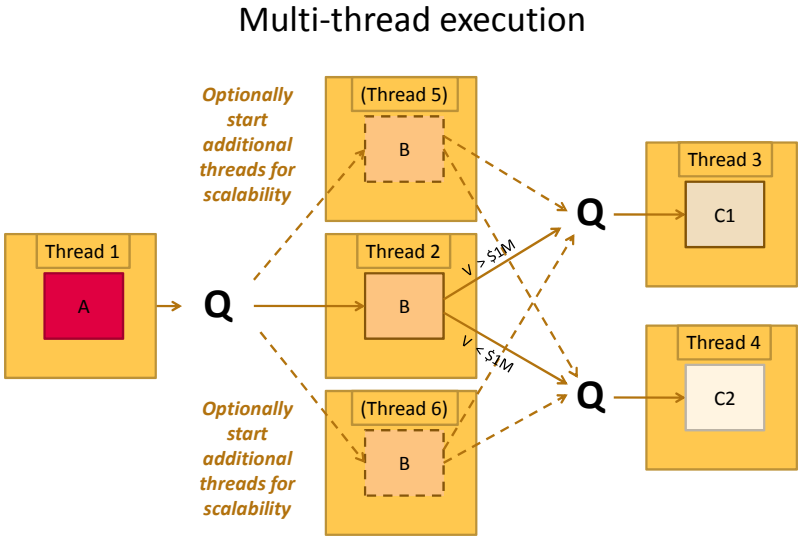

Figure 3: Message-based SOA offers loosely-coupled flexibility through multi-thread execution

This fundamental difference between the two approaches has other implications too. For instance, the loosely-coupled, multi-threaded approach offers a greater degree of tolerance to partial unavailability. If there are network problems, or a particular platform fails, this does not necessarily stop the other parts of the operation from executing, and once the failure is corrected the rest will happen automatically since the work will simply be queued up. Contrast this with the process-driven approach where a failure causes the operation to at least stop, if not fail altogether. Other effects of the two different approaches will be considered later in this paper as part of the overall discussion of which approach suits which scenario best.

Message-Driven SOA Functionality

Having discussed the message-driven SOA concept, it is now possible to look at the infrastructure requirements to support it. These are the features and functions that will be needed in any set of products and tools that are to be used to deliver message-driven SOA support. First, the fundamental requirements will be considered; those functions that are absolutely required for any message-driven SOA implementation. Then a range of best-of-breed characteristics will be addressed, representing other functions that could be of potential value to any company adopting message-driven SOA. The key is for the message-driven SOA adopter to choose an infrastructure / tools supplier that provides whichever functions provide the best match with requirements.

Fundamental requirements to support message-driven SOA

At its heart, SOA makes use of a multi-platform, multi-environment communications pipe – usually an enterprise service bus (ESB). The ESB provides asynchronous messaging capabilities between different platforms, application environments and IT technologies. Ever since the advent of asynchronous messaging and the emergence of enterprise application integration (EAI), companies have become increasingly convinced of the advantages of loosely coupling IT components together through this type of asynchronous messaging. The lack of any form of blocking, as found in synchronous solutions, enables a high degree of parallelism, and the decoupling and breaking down of synchronous ties between components makes change easier and less risky.

So, the SOA messaging backbone provides the ideal basis for message-driven SOA. However, in order to achieve the desired loosely-coupled connectivity and multi-threading flexibility between these reusable components, and to deliver some of the key advantages of SOA, various mediation functions are required within the communications pipe that add value to the connectivity it offers. For a start, message transformation is an essential function. This offers the ability to define maps between different message formats to bridge between the expectations of different components. Thus, one component sends a message using its own format usage, while the receiving component actually ends up with the message in the format it expects.

Intelligent routing is also essential, as can be seen from the flow diagram depicted in Figure 3 above. This allows dynamic decisions to be taken on where to route a message next, based on a number of pre-defined factors. These factors may take into account message content, or the context of the message itself. So, for example, in Figure 3 the decision of which of the C1 and C2 components to drive is based on whether a particular value such as the value of the deal being processed is less than or greater than $1M.

Fortunately, these mediation functions are provided as a matter of course by enterprise service buses (ESBs), and therefore ESBs provide an excellent choice for the connectivity needs of message-driven SOA. But there is more required from the messaging backbone, beyond these two mediation functions, to deliver some of the key values of message-driven SOA. Once and once only message delivery, where users can rely on the knowledge that each message will eventually be delivered, without duplicates, will be an important feature. This is because the loosely-coupled, multi-threaded nature of message-driven SOA raises the possibility of different steps of the operation executing at different times (for example in the case of the failure of one platform), and in order to preserve operational integrity it is critical to be able to rely on the other steps being carried out when

possible, and not being carried out twice by accident. It is this feature that drives a lot of the scalability, fault-tolerance and reliability values of message-driven SOA.

Other fundamental requirements for message-driven SOA relate closely to overall SOA objectives. In order for a message-driven approach to be classed as message-driven SOA, it needs to deliver on a key SOA concept – that of reusable IT components. Reusability is a key driver for SOA, promising reduced development costs, faster time to market and higher quality of service delivery over time, and therefore providing support for **reusable SOA services** is an essential component of any message-driven SOA implementation. But just supporting reusable SOA services is not, in itself, sufficient. The decision to adopt a message-driven SOA approach may be a tactical response to immediate needs, but often this decision is taken against a backdrop of an overall SOA strategic goal encompassing process-based operations and everything else. On this basis, it is vital that message-driven SOA is a step on the SOA journey as opposed to a move in another direction, and therefore whatever mechanism is used to build the message-driven SOA services must also allow these services to be **exposed for use in the wider SOA deployment**. For example, the ability to expose them as web services may be the most commonly desired option.

Finally, the usual **assortment of development and management tools** will be required, at least at some basic level. On the development side, it will be necessary to provide facilities to help users message-enable the relevant programs and applications, as well as to configure the desired component flows and mediations. Testing and debugging tools will also help. From the systems management point of view, the most critical tooling will need to cover the deployment of the components and flows as well as basic functionality in areas such as security, administration, configuration, application enablement, and monitoring.

Message-driven SOA – fundamental requirements	
Messaging backbone	■ Message transformation ■ Intelligent routing ■ Once and once only delivery
SOA reuse support	■ Reusable components ■ Ability to expose components for wider SOA usage
Basic development and management toolset	■ Administration ■ Basic configuration and deployment ■ Application enablement ■ Availability status

Figure 4: Basic requirements for message-driven SOA

Given the desire in message-driven SOA to keep things as simple as possible, this overall list of functionality forms a good 'lowest common denominator' checklist that any supplier of message-driven SOA infrastructure and tools must offer.

Best-of-breed characteristics

Beyond the functionality required by all message-driven SOA projects, there are numerous other value-add functions that may or may not be relevant to individual project needs. Users should consider each function in the light of their own particular circumstances. The following characteristics will be discussed in this section:

- Events handling
- Advanced development and configuration environment
- Advanced systems management

- Performance / scalability
- Multi-platform support
- Deployment, Versioning and Lifecycle management
- Customization
- Standards adoption

Events handling

Perhaps the most important of all the value-add features for message-driven SOA is the provision of some sort of events handling support. Experience with asynchronous messaging implementation over the past decade has shown that while some activities may be driven by users, a high degree of automation is possible if actions can be triggered based on some sort of event occurring. At the simplest level, for example, the application to process a particular queue of work might run most efficiently if it only bursts into action when there are at least ten items on the queue.

Arguments for events-based operations have become common recently, centred around the event-driven architecture (EDA) concept, and this events handling concept is particularly useful in message-driven SOA projects. The fundamental reason for this goes back to the basic structure of message-based SOA flows as illustrated in diagram 3 above. The point is that communications in a message-driven SOA are asynchronous, bounded by individual queues. In simple terms, components pick work up off one queue, and place the output on another. Therefore, these components are ideally placed to respond to work items no matter how they are raised. Messages may arrive on their input queues as part of a flow resulting from an application request, or as the response to a particular situation occurring – the component does not need to know what the circumstances are, it just processes the message. So message-driven SOA is naturally suited to the use of events.

Events have a particular application in an SOA environment, however. Remember that reusable SOA components represent individual business functions, and that a big benefit of SOA is to get better business and IT alignment. Message-driven users may well want to avoid long, time-consuming, process analysis and reengineering efforts, but that doesn't stop them wanting to try to improve the tie between business and IT where possible. Defining an event, however, is all about specifying the conditions that must be met for that event to have occurred, and then describing the corresponding action to be taken – typically kicking off one or more message flows to drive the required response. At its most basic, this is like specifying business rules that govern how operations should be changed in the light of particular business occurrences, just as one would do with a Business Rules Engine. This is a gross simplification, of course, but events provide a simple way to garner some of the benefits of rules-based operational control without having to go to the extreme of full BPM and BRE implementation.

There are actually two major types of events here; system-generated events and application-generated events. These refer to the different ways the determination is made that an event has occurred, which then in turn triggers the subsequent pre-defined message flows. System-generated events reflect the identification at the system level of the required conditions being met, such as a queue growing beyond a certain size or a response time moving out of acceptable boundaries. These events are detected and actioned by the message-driven SOA infrastructure. Application-generated events are those events that are detected by an application. In this case, the application raises the event directly so that the infrastructure can take appropriate action. Typically the more business-related events, such as referenced in the discussion above on business rules, are raised by application components since they usually have a better understanding of business context, while system-generated events are often responses to some technical occurrence.

Advanced development and configuration environment

Another important area to consider when evaluating potential message-driven SOA solutions is the interface for the development and configuration of the SOA components themselves and the required linkages between them, together with any event definitions. Since one of the aims of message-driven SOA is to keep things as

simple as possible, the area of implementation is critical. Tools offering intuitive, interactive, graphical interfaces are likely to be the most attractive.

The first area of focus is tools provided to help 'message-enable' the base applications and programs. The problem being addressed here is that in order to participate in a message-driven flow, a piece of code must be engineered to be able to receive input messages and transmit output messages appropriately. A basic approach to this challenge will be largely program-based, with the user having to manually message-enable the desired components, but obviously more advanced message-driven SOA tools might offer additional assistance. One example is the wrapper concept, where an existing program is 'wrapped' by a layer of software that works with the existing input and output mechanisms to translate these to messages transparently. Another is the bridge concept, where bridges are offered into particular environments to offer a similar level of transparency. In the IBM mainframe scenario for instance, a bridge might provide the linkage between the desired messages and the CICS COMMAREA which provides the interface to CICS applications. A third option would be adapters for packaged applications such as SAP or SIEBEL, which can translate messages into package services and back.

Once a program has been message-enabled, it can now be fitted into a message-driven SOA flow. Ideally, it should be possible to draw the required flows graphically, and drag and drop reusable SOA components into the flow as needed. Wizards will definitely be helpful here. Then there are a range of other activities to be carried out, such as defining events to be utilized by the event handling facility, creating the queues that transfer information between services and building any mediation services required in the flow such as message transformations. The interface to carry out these activities could be keyed off the flow diagrams already assembled, where event conditions and mediation services could be specified in their appropriate place within the flows. Indeed, as far as possible it is beneficial to provide all these development and configuration activities in the same tool-based environment to avoid users having to keep switching tools and interfaces. Anything that can be done to simplify these development tasks will be beneficial.

Another aspect of the advanced level of development and configuration support to consider is how components are catalogued and discovered. In order for components to be reused in the future, developers need to be able to quickly find and identify suitable candidates. Therefore, an interface must be provided that not only indexes the available components but also explains what they are, what they do and the message-driven requirements to operate them. If reuse is to succeed, it will be imperative that this tool offers an easy-to-use, browse-based interface or developers will just ignore it and write everything themselves from scratch.

Advanced systems management

Although a certain amount of administration and monitoring is included in the fundamental level of functionality, some users may want more powerful tools in this area. For example, monitoring at the component level of a particular message-driven SOA flow might not just show whether components are working or not, but also give statistics on volumes and response times. A drill-down capability would increase value even further, enabling users to quickly identify the cause of problems and resolve them accordingly.

Tracing and debugging functions will also be very valuable, especially since in an asynchronous messaging environment it can be very difficult to understand what each component is doing and for whom. Whereas in a synchronous model, there is a direct application thread back to the driver of the activity, in loosely coupled asynchronous messaging systems components will be reacting to a particular message, with no obvious knowledge of the sender. One specific area important in message-driven operations is to have a tool to analyse the messages. Since a lot of information used by the infrastructure may be locked into the message header, or envelope, this tool will be needed to translate what is there into information that has value to the technician. This type of tool is extremely useful in failure scenarios, where messages may have become stuck in a queue, and can also be really helpful in testing scenarios.

Security is another big area for message-driven SOA. Obviously, a basic level of security has to be provided in any message-driven SOA infrastructure, since the idea of components being driven by anyone putting a message on its input queue can seem frightening to some. However, at the more advanced level, in particular it may be necessary to support whatever security environment is already in place. Also, many users will want the security facilities to cover not just the operations of the message-driven SOA, but the usage of the development and configuration tools, since these could potentially be used to alter operations. In addition, related to the security area, some companies may also have strict audit requirements, and therefore an auditing facility may be needed to log such things as changes made to the system or the occurrence of particular events.

Performance and scalability

Performance and scalability is a tricky area for message-driven SOA. On the one hand, the types of operations that will benefit from message-driven SOA solutions are likely to be highly automated, with correspondingly high expectations of performance and scalability. But on the other hand, a balance has to be struck between keeping functionality simple and uncomplicated and delivering ever more sophisticated technology to squeeze out more and more performance. However, one area where a message-driven SOA infrastructure clearly can contribute, based on its multi-threaded architecture and loosely-coupled model, is parallelism. Because message-driven SOA breaks operations up into components that are triggered by messages and can run across multiple threads and/or hardware platforms as shown in diagram 3 above, then as long as the operation does not require serialization, components can be executed in parallel. This is a significant benefit in both throughput and scalability terms.

Also, because of the dynamic routing capability built into the ESB concept, and the independence of SOA components, the messages could be routed to other platforms for execution based on some sort of load-balancing algorithm. So, there are clear opportunities for value added functionality around the area of dynamic load balancing. At one level, this could be achieved through a mechanism for an operator to detect problem situations and bring additional resources to bear on the problem, but at another level this could potentially be handled automatically by the infrastructure.

Dynamic change is another key value-add area in scalability terms. As message-driven SOA deployments grow, it will become less and less acceptable to have to quiesce any part of operations in order to bring a new service online. Instead, users will look for dynamic change facilities that can bring in a new component or flow, or modify an existing one, without impacting ongoing operations.

Multi-platform support

Many users have to worry about a wide range of IT platforms, ranging from legacy mainframe and AS/400 systems through UNIX servers to Windows workstations. The process-driven SOA approach requires components in these environments to at least be turned into services that can be invoked from anywhere else, and preferably to also be able to access local orchestration support. However, this functionality depends on the provision of specific tools to handle these needs if the platform is anything other than Java or perhaps .NET.

In a message-driven environment, this support is a lot easier to provide as long as the platform supports the messaging technology. Because the interface to a component is a message, the message-driven SOA approach is essentially non-invasive in nature. Admittedly, if wrappers, bridges and adapters are needed to improve development productivity, then these may also have to be able to offer multi-platform support. But at a fundamental level the only thing really required to make message-driven SOA multiplatform is the ability to actually deliver and receive messages to and from each environment.

Deployment, Versioning and Lifecycle management

Deployment in any SOA can be a real challenge. The root of the problem is that SOA is about connecting different IT programs and components together, and therefore care has to be taken to ensure ongoing consistency. Implementing partial changes could have disastrous impacts. The loose-coupled nature of message-driven SOA helps here, because the ties between components are not as rigid as in the BPEL-defined

process case. However, care must still be taken, and the right tools can make a big difference. Users will want an interface that can show which components are already deployed and what their statuses are, and this interface must also allow the user to deploy new or modified flows as required. A key feature of value-added deployment support will be a related facility to understand how already deployed components interact with each other across the complete message-driven SOA deployment - in other words, who is making use of a particular component. The reason this is so important is that in order to understand the impact of changing a particular component or flow, it is necessary to understand in what situations it is being used. For example, choosing to remove a particular component will be disastrous if that component is actually being used by a message-based operational flow somewhere else.

Versioning is closely related to the deployment question, and also the whole area of lifecycle management. For example, assuming a user will not be allowed to close the whole system down every time a new service is to be deployed, the toolset must be able to handle multiple versions of a service running at the same time. Therefore, it must be possible to identify the version levels of different in-flight instances in order to debug potential issues or to ensure correct execution of the operational flows.

Lifecycle management relates back to the discussions over simplicity, the reusability of SOA components, and the inherent confusion when dealing with asynchronous modes of operation. In the message-driven SOA environment, support for lifecycle management could be extremely helpful. Because components in a message-driven SOA can be driven so easily, simply by delivery of a message, care must be taken when building and testing components and flows to ensure they are not unwittingly exposed to the wider production environment. Then once unit testing is complete, most companies will have stringent quality control procedures that govern how this new operational flow is deployed into the production environment, quite possibly going through a number of quality assurance phases. This process may also involve particular governance gates, such as getting the appropriate sign-off before a new service is deployed, so life-cycle support in this situation may need to interoperate with other management or library control tools.

Customization

In the message-driven SOA case, extensive customization support will be very valuable. The reason is that, as discussed previously, message-driven SOA is usually deployed in response to a bounded business need, where there is a desire to deliver the new function quickly. In this scenario, users will be looking for any short cuts that might present themselves. The most suitable infrastructures will probably be those that do not ‘get in the way’ of the developer, but that provide the appropriate exits and access points to allow developers to use the infrastructure as flexibly as possible. Rigid adherence to specific high-level interfaces may force a level of discipline and standardisation in development, but it also limits developer creativity and, in the message-driven SOA scenario, speed.

As an illustration, consider the messages used within the SOA environment. In a process-based SOA environment, these will typically be SOAP messages. In most SOA infrastructures, the contents of the SOAP message are inaccessible to the user in-flight. But it may be that the user wants to enrich the XML in the message, for example, or perhaps do some specific validation. Similarly, in a message-driven SOA environment there is a lot of information held in the message. The value-add approach to customization would allow the user to get in and access the message contents and even the header as required. Obviously, this can be a danger to the correct performance of the system, but the point is that if the user is at least offered the opportunity, then a risk-based decision can be taken to go ahead with this sort of manipulation anyway.

Standards adoption

Finally, standards are relevant to message-driven SOA, just as they are to process-driven SOA. However, the extent to which the message-driven SOA solution implements the myriad of standards that are around today in the SOA space will be largely a question of personal taste for each user. Some users will want strict standards adherence, perhaps as part of corporate policy, while other will take the view that standards are fine as long as they do not get in the way.

Some of the most relevant standards in the message-driven SOA case will be those relating to the messaging backbone, since this is at the heart of any message-driven SOA deployment. JMS is an important standard here, and would provide the user with the option of perhaps substituting a different JMS pipe in the infrastructure, perhaps to take advantage of an existing, traditional messaging deployment. From the SOA point of view, XML is very important since it is at the heart of the ability to loose-couple components. Then, as mentioned previously, if the user wants to use message-driven SOA as part of an overall strategic SOA journey, support for making components available as web services, with WSDL interfaces, will probably be required.

Message-driven SOA - Best of breed characteristics	
Events handling	■ Application-generated events ■ System-generated events
Advanced development and configuration environment	■ Ease of use ■ Message enablement o Wrappers, bridges, adapters ■ GUI flow-building tools and wizards ■ Events specification ■ Mediation services ■ Catalogue support and easy access
Advanced systems management	■ Advanced monitoring o Statistics (eg volumes, times) o Drill-down capabilities ■ Tracing / debugging ■ Message analysis ■ Security o Fit with what is there o Tools usage authorization o Audit
Performance / scalability	■ Multi-threading ■ Load balancing ■ Dynamic change
Multi-platform support	■ Messaging backbone ■ Adapters / Wrappers / Bridges
Deployment, Versioning and Lifecycle management	■ Deployment status ■ Usage cross-reference ■ Simultaneous support for multiple versions ■ Isolation of development / test from production ■ Governance of dev / deployment process
Customization	■ Flexibility ■ Access to infrastructure data
Standards adoption	■ JMS ■ Web services

Figure 5:- Best of breed characteristics for message-driven SOA

What is message-driven SOA's sweet spot?

Having discussed message-driven SOA objectives and functionality, it only remains to try to identify those areas where message-driven SOA might be the most appropriate choice – the message-driven SOA sweet spot. Clearly, as discussed previously, message-driven SOA may not be the ideal choice in every circumstance. For example, companies looking for process-based business transformation and enterprise-wide streamlining of processes will want to lead with the full, process-driven SOA model. But when is message-driven the ideal choice?

There are many different sources of SOA terminology and definitions, but one common classification is that there are two main SOA usage patterns commonly adopted:-

- SOA as a basis for **uniform design** of IT systems, where redundancy is eliminated and functionality is delivered in a broadly homogeneous fashion
- SOA in an **application integration** role, where newly developed, uniform SOA components are integrated with existing, independently-designed programs, applications and packages

In the uniform design scenario, typically the user is moving down a 'rip-and-replace' path, where existing functionality is replaced by shared SOA services which are consumed by user-facing front ends. It is this scenario where process-based design is most likely to be of interest, since it represents the ideal time to get the processes clearly specified and the IT resources lined up accordingly. As such, this scenario is of little interest from a message-based SOA point of view. In contrast, the application integration scenario by definition must encompass existing applications, and these will all have their own business logic built in. This makes it difficult to accurately draw up the business process flows and relate them to individual components. Therefore, the application integration scenario is more naturally suited to message-driven SOA, and the source of the message-driven SOA sweet spot.

Message-driven SOA for Application Integration

Having identified application integration as the most appropriate pattern for message-driven SOA usage, it is now worth delving a little deeper to provide clear guidance on the best fit. Gartner Group talks about three main flavours of application integration, namely:-

- Data consistency integration
  Keeping multiple data sources spread across different applications and locations consistent. Typically, this involves one-way transfers of data to reflect any changes
- Multi-step process integration
  Orchestrated execution of different activities, such as software, intelligent devices or human interactions, designed to execute a particular business process. In this integration style, applications usually participate as individual steps, interconnected through a series of one-way message/data flows
- Composite integration
  A mix of uniformly created services and existing applications, assembled into a logical user view of a single application and linked through a set of request-reply connections

It will come as no surprise to anyone with any memory of the EAI movement of a few years back that message-driven SOA finds its best fit in the first two integration styles; data consistency and multi-step process integration. These were typical uses of messaging in the EAI sense, and they remain so today. As for composite integration, in practical terms the tools used to assemble or 'compose' these single-view applications are closely related to the process modelling and design environment of process-driven SOA.

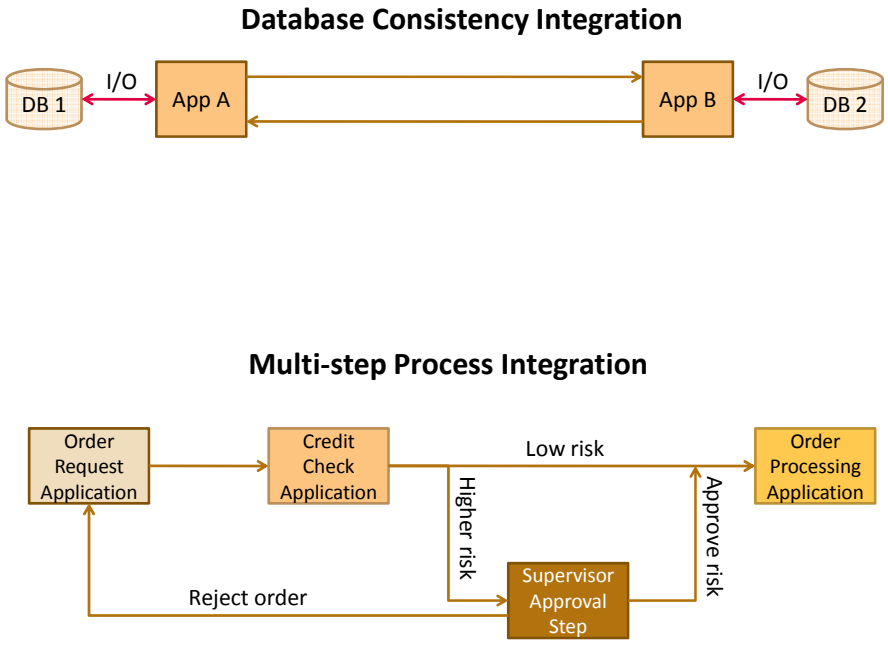

Figure 6: Data Consistency and Multi-step Process integration styles, showing information flows

The diagram above shows the data consistency and multi-step process integration styles. It is immediately clear that the database consistency integration style is a natural fit for the message-driven approach to integration – it is quite likely that there will be no real-time, synchronous requirement in this type of operation, since as long as the data in one database reflects the information in the second accurately, consistency will be maintained. A great advantage of the asynchronous messaging approach to this style of integration is that if one system is currently unavailable, the messages to trigger the required changes will simply be stored and delivered when possible. If a synchronous approach were used, the synchronization process would either have to wait until the second system was available or issue a failure response.

The multi-step process integration style is also a good fit for the message-based approach. Since each component is an existing application of package, the messaging approach requires minimal changes to achieve the required communications. There are some considerations to take into account however. The fundamental difference between process-driven and message-driven SOA is that while the former executes copies of the desired flow model, the latter executes each component in a loosely-coupled fashion, relatively independently. It is this independence, for example, that gives message-driven SOA the opportunity to deliver scalability and performance through the use of multiple threads across the different components. However, this very independence can be a challenge to managing state across each stage. This has two implications - multi-step process flows with extensive state management and serialization requirements can become quite complex, and therefore long-running applications such as those with significant human interaction may also have the same effect since they tend to drive a greater need for managing state.

So, assuming the user wants to move in an SOA direction, and is looking for the ‘quick-hit’ tactical wins for the business, the sweet spots for message-driven SOA can be summarized as:-

- Database consistency application integration scenarios
- Multi-step process application integration scenarios, particularly where
    - State management requirements are minimal
    - Interaction between components is of the short-duration, application-to-application type

Message-driven SOA sample scenario

The following example should help to illustrate the message-driven SOA sweet spot. A freight delivery company might have a number of different systems that handle freight delivery – all existing applications or packages, all different, and running across a range of platforms and locations. Some freight will be delivered by truck, others by plane then truck, and the system handling the plane side of the process is completely different to the one handling the trucks. There could even be a different system for trucks in each individual country. Now, imagine that a plane is diverted because of weather. Truck systems in the affected countries will need to be alerted that the pick-up / delivery point has changed, and it may even be necessary to organize another flight for some items that were simply using the original destination as a stop-off point on a longer journey.

This is an ideal problem to tackle with message-driven SOA. The solution would fire off messages to the affected systems in response to discovery that the plane had been re-routed, triggering a range of asynchronous activities. Trucks scheduled for pick-up at the original landing location could be rerouted or cancelled, trucks in the new location could be organized, items with other long-distance destinations could be scheduled for other flights. These tasks are all likely to be of the application-to-application type - short duration, with little or no human involvement - so a message-driven solution is ideal.

But message-driven SOA goes further than some other tactical message-driven implementation. It brings the SOA values to the table, ensuring that each of the components involved can be built into reusable services that can now be reused in other circumstances. For instance, if a carton of packages failed to make its scheduled flight, a lot of the same components would need to be triggered. Also, the linkage of the message-driven SOA components to the business operations they are implementing make it possible to start getting a clearer business-oriented picture of what is happening in at least this part of operations.

Summary

Service-oriented architecture (SOA) promises so much to companies of all different shapes and sizes, right across the world. However, the practicalities of implementing enterprise-wide SOA, reaching from the business processes down to the individual operational components, can be daunting to some. However, new approaches are emerging, striving to deliver some of the benefits of SOA without all of the work.

Message-driven SOA brings together the powerful concepts of message-driven integration with key SOA attributes such as developing reusable, business-aligned services to offer a pragmatic approach to solving business integration needs while at the same time benefiting from lower costs, quicker time to market and improved business visibility of operations. By adopting the message-driven SOA approach, companies can follow a cost-conscious, tactical needs-driven IT agenda while at the same time positioning themselves for future progression to wider strategic SOA goals.

About Lustratus Research

Lustratus Research Limited, founded in 2006, aims to deliver independent and unbiased analysis of global software technology trends for senior IT and business unit management, shedding light on the latest developments and best practices and interpreting them into business value and impact. Lustratus analysts include some of the top thought leaders worldwide in infrastructure software.

Lustratus offers a unique structure of materials, consisting of three categories—Insights, Reports and Research. Insights offer concise analysis and opinion, while Reports offer more comprehensive breadth and depth. Research documents provide the results of practical investigations and experiences. Lustratus prides itself on bringing the technical and business aspects of technology and best practices together, in order to clearly address the business impacts. Each Lustratus document is graded based on its technical or business orientation, as a guide to readers.

Terms and Conditions

Lustratus Research Limited
St. David's, 5 Elsfield Way, Oxford OX2 8EW, UK
Tel: +44 (0)1865 559040

www.lustratus.com

Ref SC/LR/92797228V1.0