



*Illustratus*

*Research*

*Best of Breed Mainframe  
SOA tools V2*

*2008 update of the popular paper  
for users wanting to bring the  
mainframe into the SOA world*



*Author: Steve Craggs  
August 2008  
Version 1.00*



*Iustratus*

## *Table of Contents*

Executive Summary .....	1
Introduction .....	1
Service-Oriented Architecture .....	2
What is SOA? .....	2
Implementing an SOA .....	3
Why SOA for the Mainframe? .....	4
Mainframe Considerations .....	5
Technology-related factors .....	6
Learning the lessons from previous mainframe SOA projects .....	6
Mainframe SOA Tools .....	9
Basic functions .....	9
Best-of-Breed Characteristics .....	9
Development / Deployment .....	10
Operations .....	13
Flexibility .....	15
Summary .....	16

# *Executive Summary*

In today's IT marketplace it is easy to think that mainframes are being marginalized more and more in companies across the world. Indeed, not so long ago the story was that the mainframe was the dinosaur, headed towards global extinction. This forecast has, of course, been thoroughly disproved, as many companies continue to gain great business benefit from their mainframe investments. Indeed, the mainframe actually seems to be staging a resurgence as people realize that its unsurpassed levels of availability, scalability and performance are still as important to business today as they were twenty years ago.

One major shift that has helped this revival is the move by IBM and the mainframe-oriented ISVs to acknowledge that value will best be preserved and leveraged by opening up the mainframe to the rest of the IT world as much as possible, moving it from inside the datacenter into a much more collaborative role. This shift has taken advantage of a timely new development called service-oriented architecture (SOA), which enables the aggregation of a diverse set of IT assets in order to deliver business services that support operations more effectively, with minimal effort and without the need for specialized skills. Essentially, programs are turned into black boxes that can be used to build business services without any knowledge of where or on what system they are running.

Companies with large mainframe investments will immediately appreciate the benefits that SOA brings. At a stroke, years of mainframe investment becomes accessible in a relatively simple way to all areas of the business, and indeed, a host of new applications also become accessible to the mainframe, allowing it to take full advantage of technological developments such as the worldwide web and Windows or UNIX environments. These different technology investment areas can feed off each other, creating maximum value and improving the return on assets.

But there is more. Turning routines into building blocks enables a much higher level of code reuse. As a result, development and maintenance costs are reduced, time-to-market and agility are improved and business risk is reduced. It is for these reasons among others that many mainframe companies now consider their SOA initiatives to be such a high priority.

However, no change is without risk. Executives of mainframe-oriented companies are often uneasy about allowing the mainframe world to merge with the rest of the IT structure. For example, concerns abound about preserving service levels and maintaining security and integrity, and there is also a general feeling that extensive retraining is going to be needed. The reality is that generic tools developed without considering special mainframe needs will not do the job effectively. Also, blindly applying SOA principles across the mainframe can prove disastrous, as early mainframe SOA users have found out. Success or failure with mainframe SOA activities will be governed, to a large extent, by the effectiveness of the tools to support this highly specialized environment, and so companies should look for toolsets that are specifically oriented to mainframe SOA development, deployment and operations.

This paper considers the topic of mainframe SOA, and identifies the best-of-breed characteristics to look for in evaluating any mainframe SOA toolset, in order to help companies make the decision that will best suit their individual needs.

## *Introduction*

In the last couple of years or so, interest in service-oriented architecture (SOA) has increased dramatically. In part, this is due to the potential benefit of turning collections of IT programs and resources into reusable, commonly accessed, business-aligned services, offering improved business agility, visibility and control while lowering costs and reducing business risk. But another contributor has been an overall increase in SOA

maturity, both in vendor offerings and by potential adopters. So, whereas when the original paper on this subject was written in 2005 the market was in its early stages, the tools on offer today are in better shape and users are clearer about what they actually need.

As might be expected with a 'hot' technology, there has been an explosion of SOA standards, particularly related to web services and XML. Although not a new concept, SOA has been propelled forward in the last year or so by the establishment of standards and technologies, such as XML, JCA and web services, that make SOA much easier to implement. These provide easy, standard ways for accessing and presenting information to heterogeneous applications, fundamental requirements in an SOA implementation. As well as making it a more practical option, these developments also widen the range of companies that can now aspire to an SOA.

For mainframe users, the independence of technology environment combined with the connectivity and accessibility offered by a service-oriented approach holds a great deal of promise. The IBM mainframe has proved to be an extremely long-lasting hit with many companies across the world, but it seems a somewhat arcane and elitist world to many IT people. Finding or training programmers to understand both the mainframe world and that of Java, Windows, .NET and web browsers can prove a major inhibitor to improved integration. However, this integration is vital if mainframe companies are to improve overall business performance by exploiting distributed technologies while leveraging past investments. SOA offers a way to mix and match mainframe and distributed assets, making it possible to ensure that overall benefit exceeds the sum of the parts.

But the integration of SOA in a mainframe environment has its own challenges. Putting together these two diverse worlds can cause all sorts of problems, from an inability of technicians to discuss their areas of expertise with one other, to the risk of compromising the high quality of service inherent in mainframe environments. Fortunately, the right mainframe SOA tools can help to address these challenges, making it easier, quicker, cheaper and safer to deploy a service-oriented architecture that encompasses both mainframe and other assets across the enterprise and beyond. This paper looks at the functionality required in such tools and provides guidance for potential buyers to assess tools from different suppliers by taking into account a range of best-of-breed qualities.

## *Service-Oriented Architecture*

Although more and more companies are becoming familiar with SOA concepts and terminology, there are plenty of opportunities for confusion. As vendors and analysts twist SOA for their own ends, this confusion is increased. Therefore, a quick recap is in order.

### *What is SOA?*

In a service-oriented architecture, IT components are related to the **business service** which they deliver, and are then made available as reusable components to be executed as part of other business services as required. So, for example, a set of programs used to gather the details about a particular customer might be assembled into a service labeled 'Find Customer Details'. This service can now be driven by any application needing to gather this customer information. This offers immediate value, since installations often have multiple copies of code used to perform the same function, and these are spread through a number of monolithic applications serving different parts of the business. A bank might have different applications for processing mortgages, insurance and account handling, each containing their own code to find customer details. In an SOA, each application would simply call the common 'Find Customer Details' service. The diagram below illustrates this reuse aspect of SOA.

### Traditional approach

- Change is difficult
- Maintenance is costly
- Brittle rather than flexible

### SOA approach

- Change is easy (once only)
- Maintenance is reduced
- Flexible rather than brittle

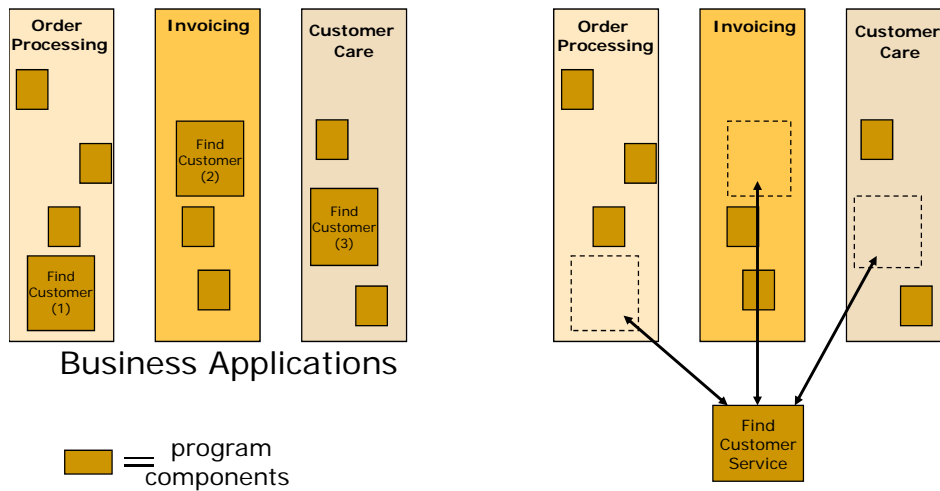


Figure 1: SOA removes redundancy and promotes reuse

It is worth noting here the source of much confusion in discussions about SOA. There is not necessarily a one-to-one relationship between business services and IT components. The 'Find Customer Details' service might invoke a combination of mainframe CICS transactions and distributed web services, for example. One might relate customer name to a customer number, another might supply the relevant address information, and a third might search out recent customer history. The whole point of working with business services is that these linkages and dependencies are hidden from the service user, preventing any use of the service from being affected by changes in these internal components and the way they interact.

The benefits derived from the reuse and flexibility offered by an SOA are extensive. Changes to business operations and processes can be implemented faster, with less cost and risk, if the IT functionality required to support the changes is delivered through the reuse of SOA services and components, limiting new code development to brand new pieces of functionality. In addition, application developers are shielded from lower level changes in the business services offered by the SOA. They can even assemble new services by combining existing ones, for example in the production of an enterprise portal. Finally, application maintenance costs are reduced by providing common services wherever possible, thus decreasing redundancy.

## Implementing an SOA

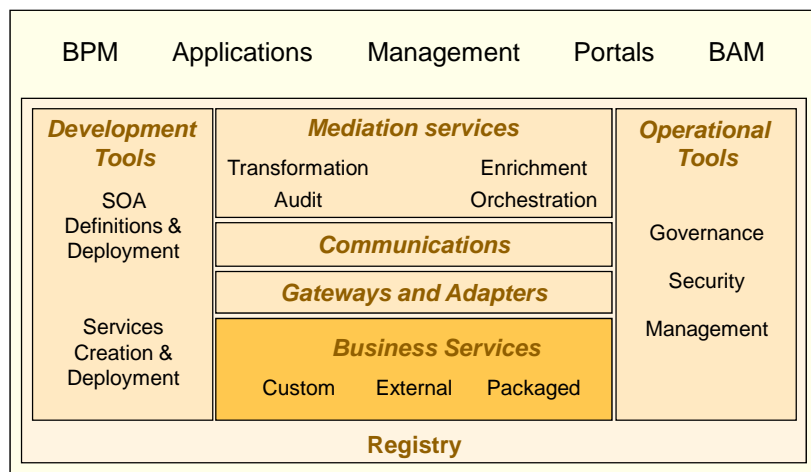
Standards such as XML, web services and JMS do a lot to make delivering an SOA easier. XML provides a way for data to be expressed by different components in a self-defining format; web services offer a standard way to access SOA components; and JMS support provides the underpinning for a viable connectivity backbone to underpin the SOA, such as an ESB (Enterprise Service Bus). Indeed, one area of confusion often seen in the market is that of web services and their relationship to SOA. Some people think that building an SOA *requires* web services while others think that web services are synonymous with SOA. Both these assertions are incorrect. On the first point, it is absolutely true that web services are very useful when developing an SOA – the technology provides a standard way to talk to and invoke a service, which is one of the basic requirements of an SOA. However, there are other ways this can be achieved, for instance by utilizing such standards as JCAs and ESBs together with a meta-data repository. On the second point, web services do not address SOA requirements like decoupling the caller from the service, the need for data transformation and reliable communications, nor a host of other SOA requirements.

An SOA 'ecosystem', far exceeding the functionality offered by web services alone, will be an essential requirement for a successful, enterprise-class SOA. Any service-producing toolsets will need to interoperate closely with the SOA ecosystem, and therefore it is worth spending a few moments considering what types of functionality will be required in a successful SOA deployment. These include:

- Service registry and governance
- Service management (e.g. Monitoring, Administration, Security)
- Transformation capabilities
- Intelligent routing (e.g. content-based, context-based)
- Communications backbone

The diagram below illustrates the make-up of an SOA ecosystem.

## SOA Ecosystem



Source: - Lustratus

*Figure 2: The SOA Ecosystem*

The registry is required to record services functionality, and the SOA should provide the ability to search and browse the registry to find the appropriate service. The metadata related to the service will also be associated with the registry entry. Service management covers the required administration and monitoring of deployed services, ensuring that the correct version of the requested service is used and that it performs acceptably. Transformation facilities will be needed when different parts of the service expect data in different formats, while intelligent routing provides the mechanism to orchestrate the delivery of the service and to apply business policies and rules. Finally, a communications backbone is clearly essential to provide the glue with which to tie the SOA together.

Having clarified what an SOA actually is, the next area to tackle within the overall theme of this paper is the applicability of SOA in a mainframe environment.

### *Why SOA for the Mainframe?*

The reality is that mainframe SOA is becoming a key topic for most major companies – indeed, IBM itself now places SOA firmly in the mainframe world as an important and relevant development.

There are a number of reasons for the appeal of SOA to mainframe-oriented companies. The benefits that attract mainframe companies can be summarized as:

- Improved return on assets
- Lower operational costs
- Faster time-to-market / increased agility
- Minimized business risk

The first point has already been touched upon. Over the years, companies have invested heavily in their mainframe environments, and financial executives in particular are keen to ensure that these investments bring the maximum possible returns. But mainframe assets in general are fairly difficult to access from the outside. There are connectivity issues, syntactic and semantic issues at the invocation level, and a huge skills chasm between mainframe and other IT staff. An SOA approach offers a way to overcome these issues. It addresses the connectivity and invocation problems, and cunningly bridges the skills chasm by enabling each skills group to concentrate on developing services. This is a key point – instead of telling a mainframe COBOL programmer that he has to work with Java, or a Java programmer that she must work with COBOL, **each person is enabled to develop services in his or her own environment.**

From a cost-efficiency point of view, SOA makes a lot of sense in a mainframe environment. Duplicate pieces of functionality are frequently found in the mainframe world, with the associated higher costs of maintenance. The ‘one service for all users’ approach of an SOA reduces these costs at a stroke. Also, if requested process changes can be satisfied, at least partially, by reusing existing services, whether on or off the mainframe, this saves in terms of project development and testing costs.

The availability of a toolkit of ‘black box’ building blocks for assembling new business services not only reduces costs but also has a major impact on delivery times. Within an SOA it becomes possible to deliver new business functionality by creating de-coupled business services that are made up of a number of existing components, where reuse is maximized and only completely new pieces need to be built. This dramatically reduces the time to market for new initiatives, delivering greater business agility. Essentially, the mainframe developer becomes a service developer, serving not just the mainframe but the wider IT installation. The challenge is to achieve this with minimal retraining, and this is one area where best-of-breed tools can help.

In terms of whether to bring an SOA approach into a mainframe environment, there are some specific potential benefits to consider. First, as reuse grows, service quality should improve too. Reusing a function avoids the danger of introducing new programming bugs, which is inherent in any code development process. Another advantage is that the ‘loosely-coupled’ approach for linking different components reduces the risk of changes to one component affecting another seemingly unrelated area. A third area of benefit stems from the predominantly asynchronous or event-driven nature of SOA implementations. In this ‘connection-independent’ mode, there is greatly increased tolerance of partial unavailability or leveling across different technologies. As an example, in a synchronous environment, each step of a process must be completed before the next can be started. Therefore, if one connection is down the process must stop. But in an asynchronous environment, different process steps can execute even if one is held up, making it possible for an informed business decision to be taken based on the information that *is* available.

All these potential benefits support the crucial importance of SOA to mainframe users.

## *Mainframe Considerations*

Before looking at the types of tools needed to effectively deploy SOA in a mainframe environment, there are a number of special mainframe-related considerations that must be taken into account. On the one hand, there are a number of technology-related factors that are either unique or particularly relevant to mainframes, and on

the other there are issues arising from the considerable body of experience built up of mainframe SOA projects over the last couple of years or so.

### *Technology-related factors*

There are four main categories of technology-related factors that users should consider when embarking on mainframe SOA projects:

- Applications and resources
- Environment
- Mainframe values
- Mainframe skills

Most mainframe applications, environments and resources are alien to those not steeped in mainframe tradition. The IBM transaction processing products, CICS and IMS, provide a complete environment in which to run high volumes of transactions, reliably and effectively. CICS is ubiquitous, used by almost all mainframe establishments, while IMS is more specialized but heavily used in the finance industry, in particular. Non-IBM products such as CA-IDMS, CA-Datcom, CA-IDEAL, Natural and Adabas are also quite common. In programming terms, COBOL is by far the most popular language, although PL/1 has its fans. The DB2 database system and the MQSeries (now WebSphereMQ) messaging middleware may be a little less inscrutable to outsiders due to their existence on non-mainframe platforms, but other system facilities such as RACF and SAF are unknown outside of the mainframe. So any toolset designed to enable mainframe SOA must be able to address the needs of these specialized mainframe applications, resources and environments.

To some extent, in a mainframe SOA deployment, technology can shield the non-mainframe world from these specialized products and environments, but the greater challenge comes in meeting expectations in terms of 'mainframe values'. Companies that rely on mainframes for much of their business have come to expect a range of benefits from their mainframe implementations. These benefits accrue particularly in the areas of reliability, robustness, scalability, performance, security, integrity and manageability. The problem is that when new applications are created in an SOA model, some components might run on the mainframe while others will run in non-mainframe environments, each with its own associated characteristics. The risk is that end-users will have come to expect a high level of service quality based on innate mainframe capabilities, and this service level could be jeopardized by influence from the distributed world. For example, while a PC user may think little of re-booting when Microsoft Word crashes, a trader might be seriously impacted by losing his dealing screen. Any tools or technology involved in building a mainframe-oriented SOA must take this factor into account.

Regarding skills, as discussed above, it is likely to be difficult and expensive to find IT developers who are comfortable programming in both mainframe and distributed environments. Therefore, any mainframe SOA toolset should enable both mainframe and non-mainframe programmers to easily and quickly create SOA-eligible components and services without the need for expensive and time-consuming re-education.

All of these mainframe environment-specific factors must be taken into account in evaluating best-of-breed mainframe SOA tools.

### *Learning the lessons from previous mainframe SOA projects*

SOA has definitely become a major consideration for a growing number of companies across the world. As discussed previously, SOA has particular attractions for mainframe users, and as such there have been plenty of mainframe SOA implementations already established. However, these projects have typically run into a range of problems, and today there is a much greater understanding of the mainframe-specific issues to take into account before embarking on a new mainframe SOA deployment. A number of the lessons learnt reflect directly back to the technology-based considerations just discussed. But one issue in particular stands out – that of **mainframe-oriented SOA service composition**.

The problem really stems from a collision between the purist world of the systems architect, and the pragmatic needs of operational service quality. Many companies look at SOA and see a pure, clean architecture where every business function becomes a service – often a web service accessed through WSDL through the use of SOAP messages – with all data presented in an XML format and orchestration carried out through the use of BPEL. This is a great ideal, but can be disastrous if implemented without due consideration, particularly in a mainframe environment. There are two major issues. The first is that, given the number of mainframe transactions in existence, this approach will result in a huge number of services being created, and experience has shown that this introduces severe management problems. In order for services to be reused, for instance, developers need to know they exist, and this discovery will be made extremely difficult if there are too many low-level services defined. Then there are governance issues, and a host of other problems. The other major issue is that this design approach is likely to be highly inefficient in a mainframe environment, where data is usually not in XML format, BPEL will introduce a highly undesirable overhead, and mainframe programmers are the only people who understand the business logic implementation.

So mainframe users will want tools that allow them to compose SOA services out of a collection of mainframe applications and data interactions, without having to worry about the overhead of SOA-dictated technologies such as XML and BPEL at every internal step. This allows services to be built that have the right level of granularity and efficiency.

In fact, it turns out that resolving this granularity issue is critical in terms of building a successful mainframe-oriented SOA implementation. For a start, correctly determining the granularity of the business service ensures the desired aim of de-coupling the service user from its implementation details, something which is vitally important when this knowledge is locked within the mainframe programming teams. Consider the ‘Get Customer Details’ example discussed earlier, currently implemented in the shape of a UNIX-based program to generate a customer number from the customer name, and two CICS transactions to look up the customer address information and trading history. An architect might create a separate business service, in SOA terms, for each CICS transaction and web service involved, thus yielding three services – ‘Find Customer Number’, ‘Find Address’ and ‘Gather Customer History’. He would then need to document the fact that the process ‘Find Customer Details’ involves invoking the first service to generate the customer number from the name, then plugging that into the second and third services to assemble the customer details.

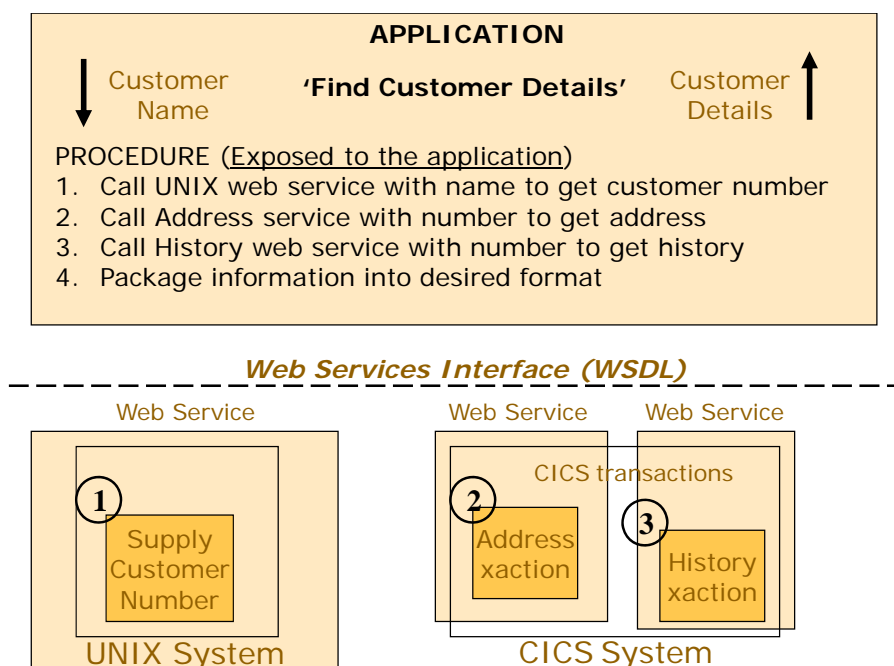


Figure 3: Poor web services granularity exposes procedural information to the application

Future users of the 'Get Customer Details' service would need to consult the documentation to understand how to use the three service calls to generate the required results, and if a subsequent change were made, the application and documentation would have to be changed to reflect the new process and invocation requirements. Contrast this approach with a more considered one, where a higher level 'Find Customer Details' service is implemented. Now, changes would not affect the caller of the service at all.

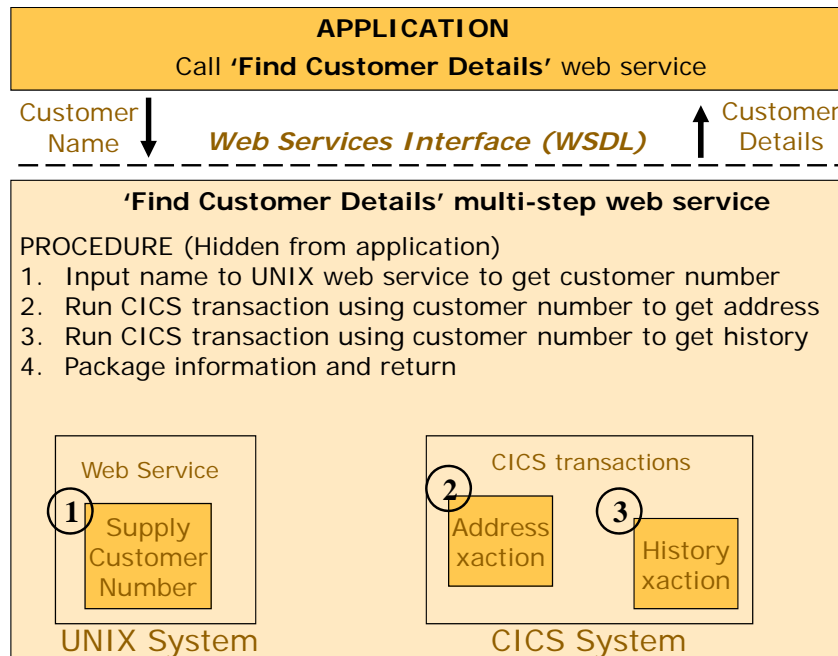


Figure 4: Getting the granularity right insulates the application

Of course, getting the business service granularity right helps with some of the other objectives of mainframe service composition, such as limiting the use of WSDL and expensive mappings from XML to mainframe data formats and back. In addition, trips to and from the mainframe are highly inefficient, and should be avoided as much as possible by optimizing traffic flow. In short, defining the optimal level of granularity:

- Decouples the service user from the implementation details of the service
- Ensures that the mainframe service directly meets the business need more closely
- Keeps the number of web services and related definitions under control
- Reduces the development effort required
- Limits the need for non-XML / XML mappings
- Optimizes performance and network load by limiting the trips to and from the mainframe

In fact, mainframe user experiences with SOA generally show that a good guideline is to avoid imposing too much of the SOA model on the mainframe environment. As commented earlier, mainframes are different to other platforms; data is often in proprietary formats, XML is almost never used, the skills set is highly specialized and expectations of performance, scalability and reliability are much higher. Therefore, the key to SOA success in mainframe environments is to implement the minimum SOA technology on the mainframe that still ensures the mainframe can become an active player in the enterprise-wide SOA implementation, and a practical approach to mainframe composition of SOA services will go a long way to helping achieve this.

# *Mainframe SOA Tools*

Having set the framework for mainframe SOA, it is now possible to look at the different requirement areas for mainframe SOA tools and to consider best-of-breed characteristics for these areas. There is a basic set of functions required to enable mainframe SOA at the purely mechanical level, and then a range of best-of-breed characteristics that can be used as a checklist to judge relevant differentiators in any tool selection. In other words, every toolset for enabling mainframe SOA has to include the basic functionality, but the support for the best-of-breed characteristics will depend on the particular vendor concerned.

## *Basic functions*

Firstly, there are certain functions that the SOA ecosystem will provide, such as a communications backbone with which to link different systems, a transformation engine to switch between different data formats and an orchestration facility to govern the flow of components in a particular service. Of course, transformation and orchestration may also be needed in the mainframe SOA toolset if multi-step processes are supported, but this is a best-of-breed rather than basic piece of functionality.

At a minimum, in order to build and deploy an SOA involving mainframe assets, the toolset must include the following functions:

- Programmable access to mainframe applications
- Screen-based access for applications with no programmable interface
- Wrappers / Adapters to provide a standard invocation interface

Essentially, this list covers the ability to get at mainframe applications, and to do so in a standard way as decreed by SOA principles. Regarding programmable access, as mentioned in the previous section the mainframe has specific application environments that control execution of online transactions. The most prevalent is IBM's CICS, used by almost all mainframe customers, and any basic mainframe SOA toolset must at the very least address the CICS transaction. Modern CICS applications are usually designed in such a way that they can be driven externally through a programmable interface, using the COMMAREA in conjunction with the LINK function to provide input to and execute the particular application. For applications that fall into this category, the mainframe SOA toolset can fairly easily enable them to be driven from outside of CICS.

However, for older applications the access problem is more difficult. These applications were designed to be run from a screen, and terminal handling is built into the application together with the business logic. Screen handling is through the manipulation of 3270 data streams. Again in CICS terms, these programs are often referred to as 'BMS Applications', that is, applications that use the CICS Basic Mapping Service facility to process screen-based menus. In order to bring these applications into the SOA fold, it is necessary for the toolset to provide a mechanism to drive them through their built-in screen-based interfaces.

Finally, an SOA model requires that components can be executed in a standard way regardless of where or on what technology platform they are running. In the mainframe case, this is achieved by providing wrappers or adapters that can bridge from the desired standard interface specification to the required mainframe-oriented access mechanisms such as COMMAREA-based LINKs.

## *Best-of-Breed Characteristics*

With this basic level of functionality, it is possible to bring at least CICS mainframe applications into an SOA. However, the task is likely to prove extremely cumbersome, error-prone, and time and resource-intensive. In addition, many companies have important mainframe applications running in other environments, such as IMS, IDMS or even batch. To address these problems, the mainframe SOA toolset will usually provide a range of other functions. These will now be considered as potential best-of-breed characteristics. It is important to recognize that the following characteristics may not all be required by every company looking to implement SOA

in a mainframe environment. Instead, the characteristics are provided as a checklist of functionality that may or may not be required. This allows a company to choose the characteristics relevant to its own requirements when assessing mainframe SOA toolsets.

Best-of-breed characteristics will be considered in three main sections:

- Development / Deployment
- Operations
- Flexibility

### *Development / Deployment*

Perhaps the first area to be discussed should be the key added value area of **web services support**. Although, as discussed earlier, web services are not *required* in an SOA, the fact is that web services offer a very convenient, standards-based access mechanism for SOA components. Web services provide standard ways in which to index a list of available services in the SOA (UDDI), to format communications between the caller and the service (SOAP), and to describe the interface to the service (WSDL). Indeed, it is likely that web services will be relatively familiar to technicians working outside of the mainframe environment, and the ability to present mainframe applications as web services will make it easy for these technicians to utilize them. The best-of-breed mainframe SOA toolset, then, will provide a mechanism to help mainframe developers bridge between mainframe and somewhat alien web services technologies.

But as discussed earlier, supporting web services is not enough. The issue of **mainframe service composition** is critical. Business services must be developed in such a way that they make optimal use of mainframe services, accessing mainframe applications and data without the imposition of unnecessary SOA technology-based overheads and have the **appropriate granularity**. Achieving the right granularity ensures that knowledge of internal process and implementation detail is decoupled from use of the service, but this is not the only reason why service architecture is critical. If web services usage is not controlled, the number of services with associated WSDL definitions and possibly XML to non-XML mappings quickly becomes unmanageable. In addition, in a mainframe environment it is important to be sensitive to local policy on mainframe resource usage. For instance, XML is very verbose, so high levels of XML activity on the mainframe may be unacceptable. Also, some companies like to keep a very tight rein on the growth of their COBOL libraries, so it may be unsatisfactory for any tool to generate or require new COBOL programs.

Another key point for a best-of-breed tool is to support the two distinct forms of designing and building services: **bottom-up** and **top-down**. These two design approaches reflect the different points of access to SOA, from the mainframe or distributed SOA worlds. Typically, the bottom-up approach involves the mainframe team looking at the mainframe assets to be exposed, considering the interfaces used such as CICS COMMAREA, and then mapping this up to the corresponding SOA services interface such as WSDL in the web services case. This WSDL can now be exported to a UDDI-compliant registry for use by service developers in the non-mainframe area. The top-down approach tends to be used when driven from outside of the mainframe. The SOA service developer defines the WSDL for the desired service, describing what the service is to do, and then passes this across to the mainframe team so that the service can be built to match requirements. Different organizations will feel most comfortable with different approaches, and therefore support is required in the mainframe SOA tool for both these design methods.

In fact, the potentially distributed nature of service development leads to another best-of-breed area of support – reference models. The idea here is that by providing developers some sort of reference template, the quality, consistency and interoperability of the resultant services will be greatly enhanced. Use of reference WSDL, for instance, provides a way of ensuring that developers build WSDL that is appropriate for the underlying programs forming the service. **XML schemas** offer a reference data model, helping to protect data integrity and ensure interoperability. This benefit does not have to be limited to the enterprise; industry-based schemas such as IFX or ACORD are designed to ensure interoperability between different companies across the industry

in question. However, in order to take advantage of these mechanisms, the best-of-breed tool will have to be able to support a **wide range of XML data types**, to make sure it can fit with whatever reference standards are required.

Following up on the coding points just made, another best-of-breed characteristic will be to **reduce coding / code generation**, or preferably eliminate it entirely. Although it is obvious that less coding will result in lower costs and faster time-to-value, it is also particularly beneficial in the mainframe case because of the fact that any coding required might well involve such non-mainframe concepts as WSDL and XML, where programmer unfamiliarity may lead to a greater potential for error. The coding burden may not seem important when the first few services are being developed, but this view is likely to change as numbers grow rapidly.

**Language support** will obviously be another key area of added value. Mainframe applications might be written in COBOL, PL/1, Assembler or even in a higher-level language like Natural. The company implementing mainframe SOA should ensure that any toolset under consideration supports the necessary languages used for its mainframe applications. This support will almost certainly need to encompass tools to map mainframe programming structures such as COBOL copybooks into standards-based formats, using technologies like XML and web services.

As mentioned earlier, the mainframe SOA toolset must be able to support various **different types of applications** such as CICS COMMAREA and screen-based ones, IMS-based ones and even batch routines. In fact, for many years there has been an active and well-developed aftermarket of vendors supplying a whole range of applications and platforms for the mainframe, and support for the appropriate ones will be an essential requirement for any user embarking on a mainframe SOA project. The toolset may also need to access **mainframe data** as well as mainframe applications, which would require access to DB2, VSAM, Adabas and other mainframe data sources, preferably under a single SQL-style interface. In addition, these mainframe resources may well need to be orchestrated with **web services** to create the overall business service. In development terms, it would be very useful to remove as much programming as possible from these activities, with the toolset handling the majority of the work automatically.

All of the best-of-breed areas discussed so far—that is, support for web services, mainframe languages and multiple application types—lead to one of the critical areas of differentiation for mainframe SOA tools – **ease-of-use**. It is quite possible for a toolset to satisfy the above requirements but still leave a lot of work for the technical staff to perform. Tools need to be intuitive, with minimal training requirements, and allow both mainframe and distributed programmers to concentrate on building services of the right level of granularity for use in both environments without the need for expensive third-party services. It is well worth verifying these facts with any prospective supplier before any acquisition is made.

A number of the previous best-of-breed areas also contribute to the other critical area of differentiation, which could be thought of as **mainframe SOA optimization**. While this may sound more like a runtime performance issue, it actually reflects back to the discussions earlier in this paper on the advantages of minimizing the mainframe's exposure to SOA. In essence, it is about bringing the appropriate level of SOA technology to the mainframe, and no more. This is partly to do with getting the service granularity right, but also about enabling mainframe programmers to contribute in service design and implementation without retraining, and not going overboard with SOA concepts where there is no need.

One interesting example here is that of the exposure of mainframe SOA services to other mainframe applications. An SOA purist might insist that if a web services interface has been developed to drive a particular business activity on the mainframe, this service interface should be used by any application wanting to invoke it. In web services terms, this would mean ensuring data was in the appropriate XML format, and then executing the WSDL through a SOAP message to drive the service. Now, while this is exactly what would happen for an external application to drive the mainframe service, this would not be desirable at all if the caller was another mainframe application. Although this might be architecturally clean, it would require XML and WSDL translation and processing at the caller and service ends where they are both in the mainframe environment – very

inefficient. So this is a clear case where forcing SOA technology on the mainframe where it is not appropriate has a damaging effect. A best-of-breed mainframe SOA tool needs to be sensitive to this issue, and provide functionality to help support the goal of only deploying SOA on the mainframe where it makes sense, and no more.

There are a host of functions that belong to the generic SOA discussion rather than the mainframe SOA tool itself – functions like service registry and management, as discussed earlier. However, although the mainframe SOA toolset does not necessarily need to provide these functions directly, it must offer **SOA ecosystem support** – simply generating WSDL is not enough. At the development/deployment level, this support should include being able to interrogate the registry to identify available services, register newly created services and provide the necessary information for the service management component to be able to understand the new services and their orchestration.

Ideally, a mainframe SOA toolset should **automate** as much of the service creation task as possible on the mainframe so that it can be done quickly, intuitively and without the need for extensive retraining. This will allow mainframe developers to quickly and efficiently create and deploy new or modified services, making the best use of available development resources.

An essential part of any development process is **testing**, and this is another important best-of-breed area for mainframe SOA toolsets. This relates partly to the skills optimization issue, but also to the nature of SOA-based services. Testing will be very much easier if different components or services can be tested in isolation, rather than having to wait for all the relevant components to be completed and assembled before any testing can occur. A test harness that enables a service developer to test a particular component, creating inputs and outputs to simulate real operations, will be invaluable in terms of reducing time-to-market for new projects and speeding up the overall development process.

Once the new services are developed, it would be advantageous if the mainframe SOA toolset provided **governance and lifecycle support**. This should allow the creation of new services and components to be controlled and managed appropriately, fitting into corporate governance procedures and then passing through development/test/QA/ production-levels to ensure that development, deployment and production operations can be managed safely. Versioning should also be supported, to allow for services to be updated in flight. Otherwise, there is an increased risk that an unprepared or incorrectly leveled change might enter production, with potentially damaging consequences.

Finally, an important aspect of a best-of-breed toolset will be the **mainframe expertise** of the tool supplier. Although this is not actually a functional requirement, this point reflects the discussion about mainframe values and the need to be sensitive to the special requirements of working in a mainframe environment. In order for the toolset to be usable, effective and efficient, it will be vital that it is created based on an extensive understanding of mainframes. For example, mainframe environments typically have stringent requirements on integrity and recoverability. There are also many operating system functions that will be useful, and in the end the 'look and feel' of the toolset will be important in order to gain acceptance within the mainframe community.

<i>Best of breed characteristic: Development / Deployment</i>	<i>Additional comments</i>
Web services support	
Mainframe-oriented service composition support	<i>Modeling support to define orchestration requirements</i>
	<i>Building the right level of service granularity</i>
	<i>Combining mainframe applications and data without imposing unnecessary SOA disciplines (eg BPEL, XML)</i>
Bottom-up and top-down service development support	
Minimal coding requirement	
Language support	<i>COBOL, PL/I, Natural, ...</i>
Support for XML Schemas and broad XML data type support	
Support for additional resources	<i>IMS, IDMS, Batch...</i>
	<i>Other native packages and systems</i>
	<i>Adabas, DB2, ...</i>
Ease-of-use	<i>Intuitive</i>
	<i>Minimal training</i>
Mainframe SOA optimization	<i>Limit SOA technology on mainframe to 'appropriate' levels</i>
SOA Ecosystem support	<i>Registry search</i>
	<i>Service registration</i>
	<i>Information to support service operation and management</i>
Automation facilities	
Repository	<i>Registry for SOA components</i>
	<i>Ability to use existing repository</i>
Asset management tools	<i>Discovery and reporting</i>
Testing tools	
Governance and lifecycle support	
Mainframe experience	<i>Understanding of 'mainframe values'</i>

Figure 5: Best of breed characteristics – Development / deployment

### *Operations*

Once the SOA services have been created and tested, focus moves onto production operations. At this stage, the SOA ecosystem becomes heavily involved with facilities such as business service management, monitoring, administration and so on. But it is important that a best-of-breed mainframe SOA toolset still maintains a close working relationship with the ecosystem. For example, the toolset needs to ensure that the ecosystem has all the necessary information from the mainframe perspective to function effectively, and there will be mainframe-specific areas where the ecosystem will almost certainly be somewhat blind.

The first area to consider is that of **orchestration**. Mainframe-oriented service composition has already been mentioned more than once as a key issue for mainframe SOA, and therefore the tool runtime will need to provide whatever orchestration is necessary to drive the components that make up the composed service. This has already been discussed at the development level, where the tool has to provide a way to build these flows, but the runtime actually implements the execution path. Ideally, this orchestration needs to support as many different mainframe environments as possible, whether they have any concept of SOA or not. For example, batch applications or vendor-supplied packages running natively on the mainframe could well be involved in

critical business operations flows, and would all need to be orchestrated as required. And of course, the orchestration support must work in conjunction with the broader SOA ecosystem runtime.

It is important that any mainframe SOA tool considers the question of **administration**, since mainframe and non-mainframe assets need to be managed seamlessly to gain the most advantage from an SOA implementation. New services need to be implemented, new users need to be authorized for allowed usage, and services being replaced may need to be closed out, to mention just a few administrative tasks that need to be addressed. In addition, SOAs often span different locations as well as environments, even spreading as far as into partner and other third party-companies. Therefore, the administration capabilities must include remote operations to encompass end-to-end transaction needs. While some of these issues fall within the responsibilities of the ecosystem, the mainframe SOA tool needs to support such activities.

Of course, **security** is another issue that will be absolutely key to many mainframe companies. Mainframe users are accustomed to a high level of security, and mainframe operations are often mission-critical in nature. These factors combine to create a high level of concern and potential risk when deploying a mainframe SOA, particularly because prized mainframe assets are now made available to the great wide world of the distributed enterprise, and potentially even outside of the enterprise to partners and other third parties. Therefore it is crucial that security is managed carefully, linking up with existing security facilities in use such as RACF. Security in this sense may need to address all four main areas of authentication, privacy (encryption), integrity and non-repudiation. Once again, this will be primarily the responsibility of the SOA ecosystem but the mainframe SOA toolset must play its part. For example, the tools themselves must be secured from unauthorized usage.

The complexities of operating an SOA model, exacerbated by the use of asynchronous and event-driven modes of operation, create a challenge in understanding what is actually happening in the live, production environment. Services may be firing off other services asynchronously, crossing platform and location boundaries at will, and it can become extremely confusing to the operations staff. The result is that it can be hard to maintain service levels and general responsiveness. The answer is for the SOA ecosystem to offer some level of **monitoring** and **problem determination** capability, with support from the mainframe SOA toolset. These facilities need to be able to glean information from within the mainframe to be combined with non-mainframe information in order to understand what is happening, and then provide the necessary investigation and action tools, such as service tracing and data flow analysis, to identify and resolve any problems spotted by the monitoring component. Ideally, this functionality should also provide some method for integrating with the enterprise management framework, for example offering an SNMP agent to alert the framework of problems within the SOA environment. While the SOA ecosystem will have a good vision of what is happening outside the mainframe, the SOA toolset itself will probably need to deliver information pertaining to the mainframe side.

<i>Best of breed characteristic: Operations</i>	<i>Additional comments</i>
<i>Runtime orchestration</i>	
<i>Administration facilities support for SOA ecosystem admin tools</i>	<i>Handling mainframe and non-mainframe environments</i>
	<i>Start / stop for mainframe services</i>
	<i>User management liaison</i>
	<i>Remote operations capability</i>
<i>Security</i>	<i>Interoperation with authentication, integrity, privacy and non-repudiation SOA ecosystem tools</i>
	<i>Protection of toolset assets</i>
<i>Monitoring and problem determination</i>	<i>Information sharing with ecosystem tools</i>
	<i>Tracking of services and information flows on the mainframe</i>
<i>Integration with existing management framework</i>	

Figure 6: Best of breed characteristics – Operations

## Flexibility

The final set of best-of-breed characteristics relates loosely to the flexibility of the toolset to support the mainframe SOA. For some companies, the ability to make the SOA truly **bi-directional** will be an important characteristic in this area. Efforts to gain more value from mainframe investments frequently focus on exposing mainframe resources for outside usage. This will certainly be the mindset of a company that is looking to stabilize mainframe investment. However, many companies have accepted that the mainframe remains a key element of strategic planning for the future. These companies are likely to be very interested in the additional benefits to be obtained by being able to operate the SOA both ways – that is, for mainframe applications to be able to run outside services as well as the other way around. This ensures that *all* IT investments are leveraged, not just mainframe ones. This bi-directional capability may require some additional work in the toolset since a way must be provided for a mainframe application to issue a request for the service.

Another aspect of flexibility relates to where SOA runtime processing is carried out. This decision will usually be made based on specific service-level requirements or restrictions. Among other things, the mainframe SOA toolset needs to deal with navigation issues for mainframe applications, such as which CICS program should be called depending on the result of the previous one, and formatting issues like dealing with 3270 data streams when accessing screen-based applications. Then there are the SOA requirements for transforming data to and from XML, handling SOAP messages and so on. Depending on issues such as performance requirements, mainframe capacity and ease of programming, it might be desirable to run all of this SOA-related processing in its own mainframe address space or within an existing mainframe environment such as CICS or IMS. Alternatively, it might be desirable for some or all of these activities to take place within a mainframe speciality engine, to reduce costs and improve performance and scalability, or even outside the mainframe altogether, perhaps in a distributed server or even an appliance system. Whatever combination works best for the user, it is clear that a best-of-breed mainframe SOA toolset will need to support this **choice of processing location**.

As just touched on, support for different processing locations and the previously discussed monitoring support are important factors in another area of consideration for best-of-breed mainframe SOA toolsets - that is, **scalability and performance**. As well as these areas, the toolset will probably need to provide some level of statistics reporting capability to allow effective capacity planning and load management. In addition, when running in a mode where most of the SOA processing is carried out on the mainframe, the mainframe SOA toolset should take advantage of native facilities in the mainframe operating system, such as cross-memory, to optimize performance.

<i>Best of breed characteristic: Flexibility</i>	<i>Additional comments</i>
<i>Bi-directional support</i>	<i>Mainframe application access to non-mainframe resources</i>
	<i>External access to mainframe resources</i>
	<i>Language stubs for mainframe program access (where needed)</i>
<i>Choice of processing location</i>	<i>Inside CICS</i>
	<i>Outside CICS but on the mainframe</i>
	<i>Making use of speciality engines</i>
	<i>Off the mainframe, on distributed servers or appliances</i>
<i>Performance / Scalability</i>	<i>Ability to choose processing location based on performance/scalability needs</i>
	<i>Statistics on services usage for capacity planning purposes</i>
<i>Use of native mainframe high-performance options</i>	

Figure 7: Best of breed characteristics – Flexibility

## *Summary*

For many companies, mainframes are set to be a fixture for the foreseeable future. As these companies struggle to deliver increased business value from their IT investments, opening up the mainframe becomes a major focus. The SOA model and its related toolsets have arrived just at the right time, offering a way not only to extend the life of the mainframe, but also to ensure that it continues to play a valued role in driving the business forwards. Breaking down the barriers between the mainframe and the rest of the IT world ensures that investments can be made wherever they make the most sense, where all of the IT installation may benefit.

But success or failure with mainframe SOA activities will be, to a large extent, governed by the effectiveness of the tools to support the mainframe SOA initiative, combined with a disciplined focus on deploying only the appropriate level of SOA technology and concepts on the mainframe. Generic tools developed without considering special mainframe needs will not do the job effectively. Instead, companies need to look for toolsets that are specifically oriented to mainframe SOA architectural, development, deployment and operations needs.

All mainframe companies will have their own specific requirements and hot buttons, and these will affect the selection of the most appropriate tools. The best-of-breed characteristics presented in this paper are intended to provide a checklist that companies can evaluate against their own requirements, and then against the toolsets offered by the various suppliers. Some of these address purely functional needs, while others have implications for other critical aspects of mainframe SOA initiatives, such as skills requirements, security and manageability. However, the goal remains to ensure that companies end up with the toolset best designed to ensure the success of their own mainframe SOA initiatives.

# *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*

© 2008—Lustratus Research Ltd.

Customers who have purchased this report individually or as part of a general access agreement, can freely copy and print this document for their internal use. Customers can also excerpt material from this document provided that they label the document as Proprietary and Confidential and add the following notice in the document: "Copyright © Lustratus Research. Used with the permission of the copyright holder". Additional reproduction of this publication in any form without prior written permission is forbidden. For information on reproduction rights and allowed usage, email [info@Lustratus.com](mailto:info@Lustratus.com).

While the information is based on best available resources, Lustratus Research Ltd disclaims all warranties as to the accuracy, completeness or adequacy of such information. Lustratus Research Ltd shall have no liability for errors, omissions or inadequacies in the information contained herein or for interpretations thereof. Opinions reflect judgment at the time and are subject to change. All trademarks appearing in this report are trademarks of their respective owners.



**Lustratus Research Limited**

St. David's, 5 Elsfield Way, Oxford OX2 8EW, UK

Tel: +44 (0)1865 559040

[www.lustratus.com](http://www.lustratus.com)

Ref SC/LR/31675249V2.0