



November 2003

Legacy Transaction Integration™ In a Service-oriented Architecture (SOA)

Introduction

Over the next year or so, every serious IT analyst and software vendor will figuratively jump upon the SOA bandwagon as the next big thing for technology. For the first time in a while, they may all be right...

So, what is a SOA and why might it be important to the way you provide service and support to your customers, be they internal employees, external partners or your customers? The key definitive word in this new acronym is the first - "Service". By Service, all are referring to the abstracted functionality or logic that performs an action – hopefully one that can be used generically by any number of users. It doesn't define the functionality per se – it only implies its universal availability.

At its core, it is a fairly straightforward and fundamental concept. A potentially complex technical function is made generic, utilitarian to all and easy to implement. Simplistically, we use the example of network printing. Users don't care – nor should they – how their data gets to the correct network printing device. They only care about results, i.e. that their Word doc, web page or PDF gets onto paper the right way. The actual technology that enables the user to select and route their data is part of the "service" provided by the printing module. Likewise, they don't care if this program is running on their system or somewhere on the other side of the network. Again, they care only about results.

If we extend this concept to a vertical business function, like others explaining this subject, we can use the ATM as a great example. The user interface is a keyboard, monitor and a plastic card, yet the "services" provided now span the spectrum of what customers can do with a teller – withdraw, deposit, check balances, transfer funds, pay bills, etc. Full function and service-oriented.

The Role of SOA's

If you extend this to its logical conclusion, Service-oriented Architectures are a conceptual way to build and deploy new technology functions that can be utilized by all classes of internal and external users across a variety of underlying systems. By all accounts, this is not a new concept, yet its serious embrace by customers will force new regimens and implementation techniques on those that deliver this functionality.

From strictly an IT internal perspective, creating an SOA means having an infrastructure in place with the ability to make “services” available for consumption by whatever user population requires this functionality. Similarly, Web Services are those functions available via the Internet that can be consumed both internally and externally. The technical delivery mechanism is comprised of SOAP, WSDL, UDDI, etc. These “enable” services to be made available via the web. Yet, they don't describe the specific services to be delivered, only the technology architectural approach. From this perspective, Web Services are the first of many ways to implement and support an SOA. Other standards-based SOA protocols in addition to Web Services include EDI, HIPAA, RosettaNet, SWIFT, ebXML, etc.

For many entirely new applications, the functionality to be delivered via SOA may need to be built. This has been the case for many of the new web applications developed and implemented via an application server. In other cases, the functionality already exists, yet is not in a format or appropriate delivery mechanism to be reused. This is true for almost all Legacy application systems, since they were initially built for an online user using a terminal.

Legacy Systems in an SOA

If 70-80% of the data used in business today still resides on legacy systems, it must also follow that the same percentage of the business logic still resides there also. This application functionality has been developed and enhanced over decades. It never breaks, is totally secure and is scaleable – i.e. deliverable - to literally thousands (...or hundreds of thousands) of actual end users.

It is our contention that these Legacy systems will continue to play a critical role in the delivery of technology solutions and will also be key components of all SOA's developed over time. In fact, Gartner predicts that legacy application functionality reuse is the “key to SOA”¹.

In our banking example, the ATM user doesn't care where the actual transactions are being processed or by what kind of system. Yet, if the truth were known, most ATM users will check the accuracy of their transaction via their web-based online banking system, as soon as they can get to a browser. If the withdrawal or debit-card purchase didn't “show up” in their account history, this SOA system wouldn't be used, because

1. “The Legacy System Continuum...”, Gartner, D. Vecchio, ebizQ Webinar 10-29-03.

it didn't completely deliver the "service" the customer wants. Today, it does and the system passes all tests, providing seamless access to the functionality and data (and cash) that the user demands.

Likewise, our network printing example must pass the test of having the correct report in the correct output tray. And, it does.

When there is both a richness and depth to the legacy applications, they will be key to the delivery of a SOA. To use a trite phrase, no one can afford to reinvent the wheel that the company has used successfully for years.

So, what is the best way to make this functionality available? There are many different ways to access the transactions of legacy systems. Your IT department can capture and wrap application functionality that is directly accessible via the applications API, or they can use technology to access the transaction directly via its execution system or they can access this logic by utilizing the user interface, via the terminal protocol and the communications data stream it supports or the browser, if it's a web application.

Which is best? If all 3 are technically available, there may be pure technology reasons why one may be better than the others. It is up to your IT department to make this decision. After all, the user doesn't care, so long as the service is provided.

Yet, in some cases, the end user access point, the terminal interface, is the best or only viable means of accessing the legacy functionality. In the past, based upon early technology implementations, this "screen scraping" approach was frowned upon by IT. With the significant advances made over the last 5 years, the technology solutions available today are based upon sophisticated data stream emulation. They are fast, reliable, secure and scaleable, encompassing all of the attributes of the legacy system itself and are therefore more than acceptable as a base for legacy SOA.

They are also relatively fast to implement and easier than ever before to maintain. And, the truth is that they may still be the only viable alternative. If your IT department won't allow new software on the legacy system host (for whatever reason) or the legacy system you need to access isn't controlled by your IT department, then accessing via data stream emulation solutions may be the only recourse. If the target functionality is a production web application external to your enterprise, then integrating this functionality through the browser may be the only option.

Presentation vs. Programmatic

There are two major technology solution groupings available today for terminal emulation. One group provides tools and execution environments for re-purposing entire existing legacy application systems. Some vendors call this rejuvenation, while others call it presentation integration. In both cases, the results are the same. The series of hierarchical screens in the original application are replaced by PC-like or browser-based

GUIs. Some sophisticated tools exist that will enable you to compact multiple screens into new consolidated views, substantially reducing the number of resulting “new” screens. If you do not require the granularity of accessing individual transactions, this is a very solid solution for entire applications.

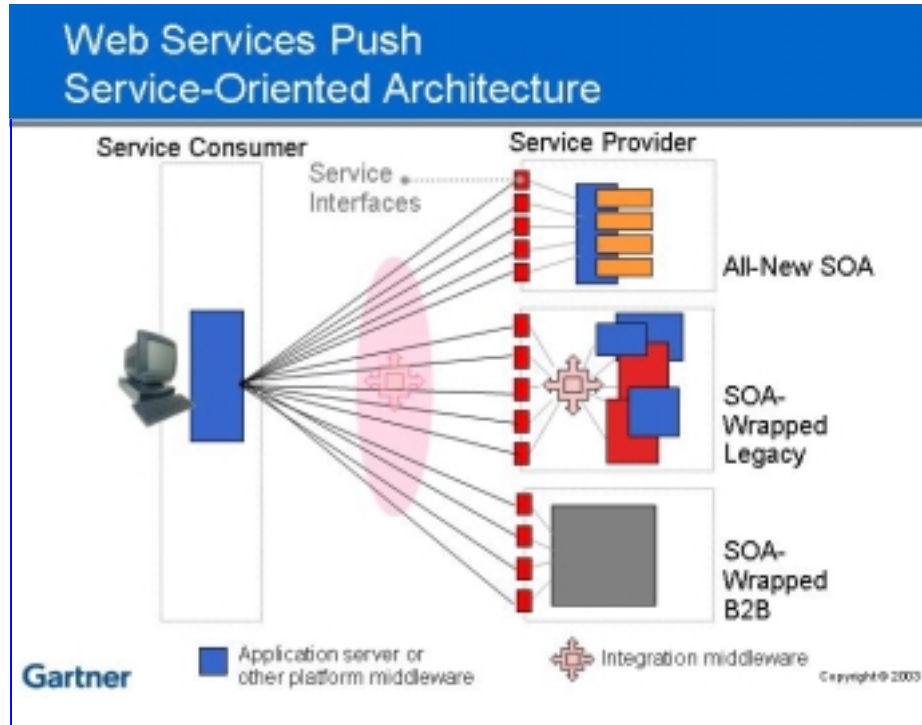


Figure 1

If, on the other hand, you need access to only specific application transactions, and these will be combined in new or composite applications, then programmatic integration may be your solution. These kinds of tools enable non-programmers and programmers alike to capture just the specific functionality of the application that is required and also enables this functionality to be wrapped and exposed as a service. It is here that legacy systems will fit best into SOA's.

Figure 1 from Gartner shows a schematic of the subsystems that they believe will be needed to implement an overall Service-oriented architecture. Integration middleware is the key to integrating legacy functionality into this new environment.

A straightforward example of legacy functionality is an inventory inquiry, where the quantity on hand can be used internally or externally in any number of new applications. The actual function of getting total quantity in inventory will likely be provided by your master inventory management legacy system. The concept in a SOA system is to capture this programmatically and make it available to any number of new applications. To return to our printing example, there's no reason to have a separate network-printing module built for each type of user or device. A single ubiquitously available software module can be made available as a service, linked where needed and executed in a SOA.

Likewise, individual transactions, some of which can be extremely complex and span tens of screens, can be wrapped and included in new multi-step applications. In fact, most tools include significant capabilities to wrap and concatenate transactions from multiple hosts – of the same kind or different – for use in these new applications. In addition, new composite applications can be built using only those specific mainframe or legacy transactions that are critical to the service required.

Gartner maintains, and rightly so, that the role of legacy transactions actually decreases as the sophistication of the new usage goes up. In essence, the inventory inquiry becomes a smaller part of a new larger composite application. Yet, they make the equally valid point that the actual value of the legacy transaction dramatically increases. When critical real time business functionality is reused and embedded in a series of new applications, it becomes extremely valuable for the critical role it normally plays. Since all legacy transactions can now be reused in any number of ways using one of the existing technologies, they become the key underpinning to the new Service-oriented Architectures that will be built in the coming years.

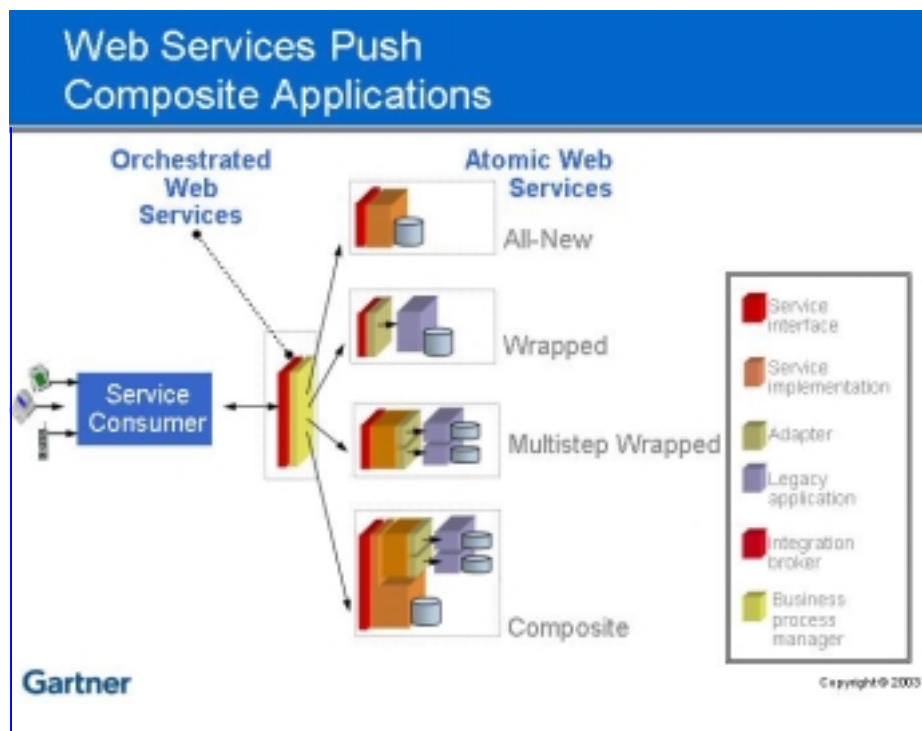


Figure 2

As depicted above in Figure 2, Web Services become the SOA delivery mechanism, with the services themselves being comprised of both new and legacy transactions. The results can span completely new applications, single legacy applications, multiple legacy applications or legacy applications as part of a new composite app. All then are linked via a Web Services “server” and made available to all users, internal and external.

This chart also covers the role legacy transactions will likely play in the SOA's, where an enterprise may implement all four types of what Gartner calls Atomic Web Services.

How do we get there?

The issue facing all companies is where to start and how exactly to chart out a course to follow. There's no overall agreement yet on a list of common best practices, but it appears that they will start to be published within the next twelve months. In the meantime, early adopters are left with nothing other than to pick those mission-critical production applications that best lend themselves to re-use.

Internal mainframe applications might include Payroll, Accounts Payable and Accounts Receivable. As mentioned above, Inventory is also a classic prime candidate. The key is to determine those application functions that can be easily captured, wrapped and made available to other users both within and without the enterprise.

It's not surprising, as a result, that security is the number one issue and concern regarding the deployment of Web Services. Knowing this, almost every Web Services software vendor has built significant capabilities to address this issue, making the customer comfortable that the potential ROI of SOA doesn't come with the unwanted side effect of security exposure.

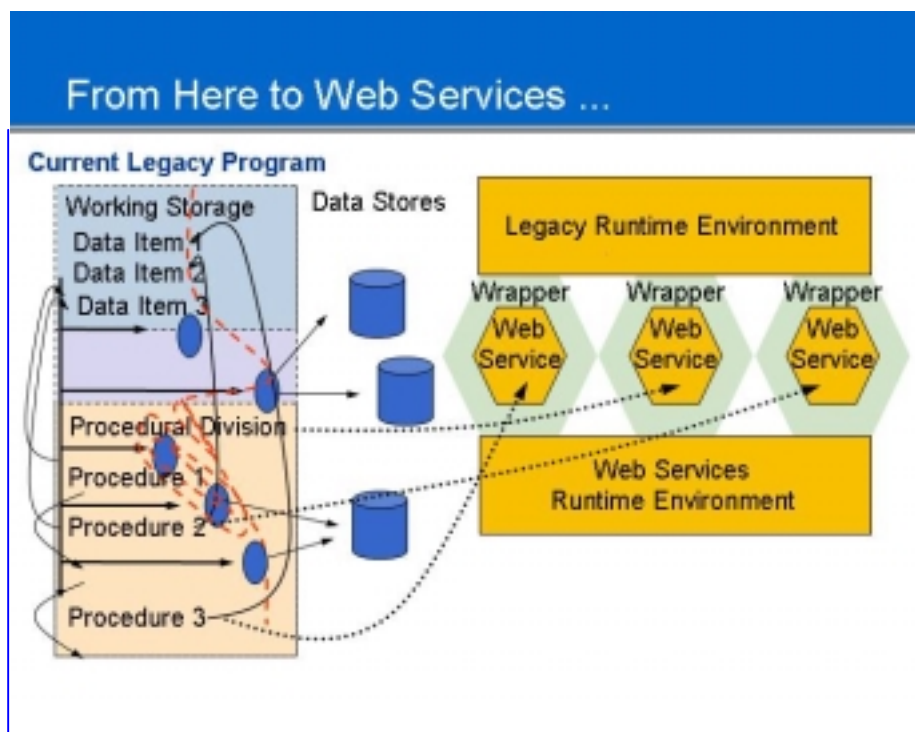


Figure 3

As depicted in Figure 3, also from Gartner, any legacy integration depends heavily on the level of granularity required. The graphic is meant to convey the varying levels of legacy application functionality that may be required, depending upon whether the need is based upon access to data, application logic or both.

By definition, Inquiries may not need access to the logic of the application. Only when this functionality is for culling the appropriate records for the exact data needed, does it become important. On the other hand, any and all Updates would normally be performed by the application, since the logic is there to insure that new data is clean and accurate before the update completes. So, depending on the need, the level of access and the need to wrap functionality can change.

There are a number of tools that provide support for both granularity and complexity. And, most of these will enable a plug-and-play interaction with all of the standard Web Services publishing systems available with Application Servers or Integration Brokers.

This portion of the puzzle of creating Service-oriented Architectures has been solved and promises to get more functional and easier to implement as time goes by. SOA's are here to stay and will only be distinguished by the level of deployment within an enterprise. And, legacy system reuse, comprised of both traditional as well as a growing number of web applications, will be a key factor in their ultimate success!