Integrate legacy systems into your SOA initiative
Convert legacy applications into SOA-based applications

Shantanu Bhattacharya

December 20, 2007

You want to include Service-Oriented Architecture (SOA) in your organization to make processes flexible, adaptable, and supple. But you already have a set of systems in use for your business processes. The solution? Integrate SOA with your legacy applications to get more value out of them. This article takes you through the steps you need to make it happen—and the pitfalls to avoid.

Why you need to incorporate legacy systems into SOA

Organizations with legacy systems see an increasing need to drive SOA adoption to derive great value out of their legacy systems. Legacy systems provide the most value-generating business processes, because they were automated earlier than the other modern systems. Most of the automation drives can't take advantage of legacy systems effectively, because these systems are more ancient than the automation experts would like them to be for applying their business value in the automation initiatives. Therefore, any strategy to bring in value without incorporating the legacy systems is bound to have limited success.

Legacy systems that started as the focus areas for their respective projects gradually lost their appeal as sources of continued system development, because the conventional wisdom was that you would be only adding to old technologies. Considering that a generational change in computing technology takes as little as three years, the problem described here is happening at a pronounced pace.

But it's clear that your legacy systems form the base from which your enterprise can continue to improve upon existing processes. So your business strategy for change in technology should incorporate legacy systems.

Related content

- Embracing SOA for the legacy world

This article categorizes the various legacy systems that can potentially be found in an organization based on the following criteria:
• What is the runtime environment they need to run on?
• For which platforms are they developed?
• Which user interface environment do they use?

A legacy system might need a runtime environment for it to run like Oracle Frames, or it might run on its own on the operating system. The legacy systems that are capable of directly running on the operating system are easier to take into the SOA fold. One such example is systems that are developed on DOS or mainframe systems. Therefore, it's a good idea to carefully choose the language and technology that you use.

Legacy applications can be of several types based on the user interface they use, of which there are:

• Green-screen applications.
• Non-GUI-based applications.
• Client/server-based GUI applications.
• Web-based applications.

Of all these, Web-based applications and client/server-based GUI applications are easiest to deal with when integrating with SOA-based applications because of their inherent nature. These applications have clear distinctions between the service provider and the UI, which is easy to turn into a service for an SOA. But that doesn't mean other applications can't be taken into the SOA fold; but these systems need massaging before they can be made to look like a service of an SOA.

Although various kinds of legacy applications have been noted here, the rest of this article details more generic issues that are encountered when transforming the legacy applications into SOA-enabled ones. Separate treatment for the various kinds of legacy application is outside the scope of this article.

**Identify which legacy systems you should incorporate into an SOA**

It's important to note the challenges of integrating legacy systems with an SOA, and that SOA-enabling all applications in a single phase would be a mistake. That would make the phase more complex, considerably reducing the likelihood of success.

To identify which legacy applications should be SOA-enabled:

• Categorize all the legacy application in terms of the business returns they provide and maintenance costs the organization would have to incur to sustain it.
• Consider the opportunity cost: What is the untapped potential of the application due to not SOA-enabling it?

The applications that are high in business value should be the first ones to be taken under the SOA fold. Also, the ones that are high on untapped potential are good candidates for transformation.
Identify the services and business benefits

After identifying the candidate legacy applications, it's equally important to identify the candidates for services. This determines the amount of success you can get from the whole transformation exercise.

This is a two-pronged process: Phase one consists of identifying what can be defined as service. Phase two deals with the business benefits you can realize from forming some services versus others.

Phase 1: Identify services

While identifying a business process as a service, you should check the following:

- The number of input and output parameters should not be very high. If it's high, then you can typically break the business process into multiple services. The reason for this is simple: After service formation, the conversion of parameters to and from the native format can take a considerable amount of time.
- The service should be doing a single function. Keeping the service as a small, granular activity takes care of this problem and makes it easily usable in many more scenarios.
- Each service should have a business function and not a technical function. You should strictly avoid a technical service with no corresponding business process. This is because a technical service isn't easily understood by the business analyst. One of the most essential purposes of SOA is to make the services understandable to the business analysts.

As an example of identifying potential services, consider an order entry application whose basic operations might be:

- Confirming the credibility of the customer.
- Reducing the stock.
- Billing the customer.
- Handling back orders.

Each of the above actions can be a potential service. However, it's important to note that the elementary business operations, such as reducing the stock, can be reused directly in another context. But billing the customer is too complex of an operation, so it must be further broken down into:

- Aggregating the billing items.
- Computing the sales tax.
- Getting the customer address data.
- Producing the bill.
- Dispatching the bill.

Phase 2: Choose services that drive more business value in an SOA

The second phase consists of choosing the services that would make more sense in terms of business value in an SOA environment. The criteria for choosing the services are similar to the
criteria for choosing the candidate applications, with some differences. The chosen services should have a high value coefficient. The value coefficient of a potential service can be defined as:

\[
\text{(Business Value from the service} – \text{Maintenance Cost}) / \text{Cost of Replacement}
\]

Therefore, services with the greatest business value, least business cost, and least cost of replacement are the best candidates for incorporation as a service into the initiative. For example, if a piece of code for a service is spread across multiple modules and files, then it incurs a lot of maintenance cost.

You can categorize existing software components by language, purpose, type, and criticality. Calculating an item's value is based on analysis of the development costs, the maintenance costs, the estimated replacement costs, and the annual business value contributed by that item.

**Salvage the legacy code**

One of the first steps in this process is to salvage (for reuse) the code from legacy applications. To salvage code from an existing legacy code base, you first have to locate that code and determine its worthiness for reuse. It's not much of a problem to analyze and evaluate the code spread across a few small programs; any programmer familiar with the code can do this using a comfortable text editor. It's a completely different task to analyze several hundred large programs in search of a few reusable blocks of code. Here, too, a domain expert is required, but he or she must be supported by automated reverse engineering tools, such as Refine/C, Imagix4D, Sniff+, and Rigi.

One of the most important needs in salvaging the legacy code is to identify the business processes that the code represents. The easy way of discovering the business operations is through the results they produce. In other words, you have to identify the business process by the outcome of the legacy code executed. This helps put a finger on the business process without an elaborate effort. By identifying the variables returned by the functions processing the business operations, you can also identify the functions. If the programs are structured so that the business functions are assigned to one code block, such as a function in C or a paragraph in COBOL, then this task is simple—but that's seldom the case. It's more likely that a business function will be found scattered across several blocks of code in several modules. On the other hand, one block of code may be processing several business functions. So there is a many-to-many relationship between code blocks and business operations.

By performing a data flow analysis based on the final results, it's possible to trace the result back through all of the statements that contributed to producing it. After the statements are identified, then you can locate in which code units—procedures, paragraphs, and so on—they're in. Only those units are then copied from the original source with the variables they refer to. This technique is popularly known as **code stripping**.

After identifying the code of a business operation, the next step is to extract that code and reassemble it as a separate module with its own interface. You do this by copying the impacted code units into a common framework and placing all of the data objects they refer to into a common data interface. In C the interfaces are parameters of the type structure, and in COBOL
the objects are level 1 items in the linkage section. The end result is, in all cases, a subroutine with a call interface. The original input arguments are input parameters, and the original output arguments are output parameters. In this respect, the business logic code has been disconnected from the original user interface and made into a self-contained subprogram. This is a prerequisite to wrapping it.

**Wrap the salvaged code**

After the code corresponding to a business operation has been located, documented, and found worthy of reuse, the next step it to wrap it. The objective of the wrapping process is to come up with a Web Services Description Language (WSDL) interface for the component extracted from the legacy code. The methodology requires transforming each entry into a method and transforming each parameter into an XML data element. The data structures become complex elements with one or more subelements. The methods have their arguments and results as references to the data element descriptions. Both the methods and the parameters are built into an XML schema.

There are several tools available in the market that automate this transformation for COBOL and C/C++. Besides creating the WSDL interface description, they also enhance the wrapped component with two additional modules:

- One module is for parsing the incoming message and extracting the data from it. The extracted values are then assigned to the corresponding arguments in the wrapped component.
- The other module is for creating the returned message from the results produced by the wrapped component. In this way, an elementary operation can be reused as a Web service without having to change the code.

The two generated subroutines act as a bridge between the WSDL interface and the call interface of the original code.

**Link services into business process**

The third and final step in creating Web services from legacy code is to link the Web services to the overlying business processes. You do this by means of a proxy component. The business process actually invokes the proxy, which is available in the same address space as the process definition. As in CORBA, the proxy checks the parameters and generates the WSDL interface, which is then dispatched by some message service, such as IBM® WebSphere® MQ, to the application server.

On the application server there’s a scheduler that receives the incoming message, determines which Web service is to be performed, and forwards the WSDL contents to that particular service—in this case, the wrapped legacy code. The wrapper of the code parses the XML input data and moves the values to the appropriate addresses in the wrapped component.

After the wrapped component has been executed, its result is transformed by the wrapper into an XML output data structure, which goes back to the scheduler to be transmitted back to the Web
client. In this way, the business process can be executed on any client anywhere, and still it's able to access the legacy functions on the original application server.

**Conclusion**

Wrapping legacy applications to be made available to the organizational SOA initiative is an important part of the new push to SOA-enable enterprises. If a proper process is followed for enabling SOA, then the task can be clean and without error. This article has tried to help you understand the process and make it as easy and as streamlined as possible.
Related topic

- The IBM SOA Web site offers an overview of SOA and how IBM can help you get there.

© Copyright IBM Corporation 2007
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)