

The SOA Magazine

Feature Article



Finding Services in the Mainframe

by Mike Oara

Published: May 3, 2007 (SOA Magazine Issue VII: May 2007, Copyright © 2007)

[Download this article as a PDF document.](#)

Abstract: It is an accepted fact that legacy applications implicitly contain services which may be exposed to the outside world. However, the identification of such services is not a trivial task. A number of questions must be answered. Should one begin by deriving services from the legacy application, or rather start with all potential services and decide which would fit actual business requirements? Also, if certain services are required, what is the best way to find them within a typical legacy application? And, what is the most suitable degree of granularity at which they should be defined? This article answers such questions and develops a rational and systematic approach for locating and extracting services from legacy environments in support of SOA.

Introduction

One of the promising strategic advantages to SOA and service-oriented computing is that it empowers organizations to continue reusing and getting value from their existing legacy assets. As substantial amounts of data reside on databases implemented on large mainframes, legacy applications already have the knowledge of where the data is, how to retrieve or manipulate it efficiently, and how to combine it with special business knowledge. Completely new applications can be built to leverage legacy applications by consuming services that encapsulate and expose legacy logic.

These potential services, however, need to first be found. We use the term “potential services” to refer to application artifacts which alone or combined have all the characteristics of services. They are therefore essentially services, even if in the old mainframe paradigm they were not viewed as such.

Before starting this search, a fundamental question needs to be asked: Should one define the proper services and then try to find where they are - or - should one first find all potential services and then decide which are useful?

Approaches to Service Identification

There are three common methods used to identify services, each of which relates to an overall project delivery approach. The first, known as the top-down approach (Figure 1), advocates the up-front definition of services by architects and analysts. This group defines the best set of services that could possibly satisfy all business requirements.

While such services would be ideally suited to respond to all requirements, they may remain just an ideal, as the mainframe application may not be able to offer them. An architect may determine, for example, that a service should allow an application to retrieve the top three most expensive items ordered by a customer. Although the information is in the database, the mainframe program may be incapable of retrieving order data in this way or with this criterion.

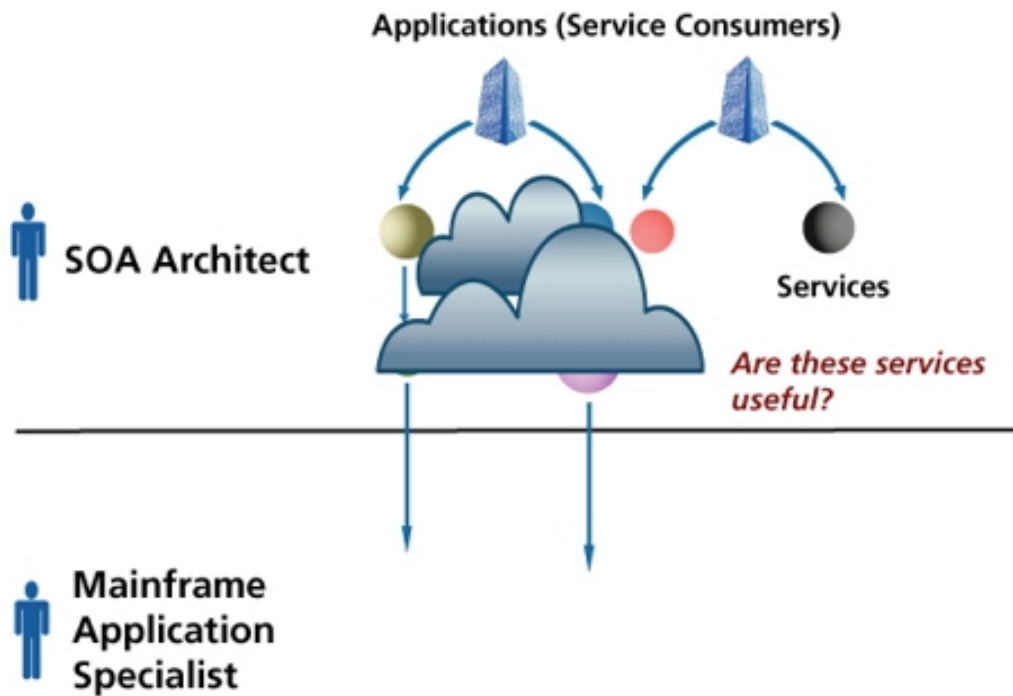


Figure 1: In the top-down approach, architects and analysts derive services from business models and then take what they can from existing legacy resources.

With a bottom-up approach (Figure 2) the definition of services originates in the knowledge of the legacy application. The services are discovered, rather than defined as requirements. The initiative belongs to the legacy application expert rather than to the service modeling team. Such an approach can be more practical, as it results in services already supported by existing legacy application functionality. However, this form of service definition is known to produce sets of services that are awkwardly defined, superfluous, and out of alignment with real business and architectural requirements.

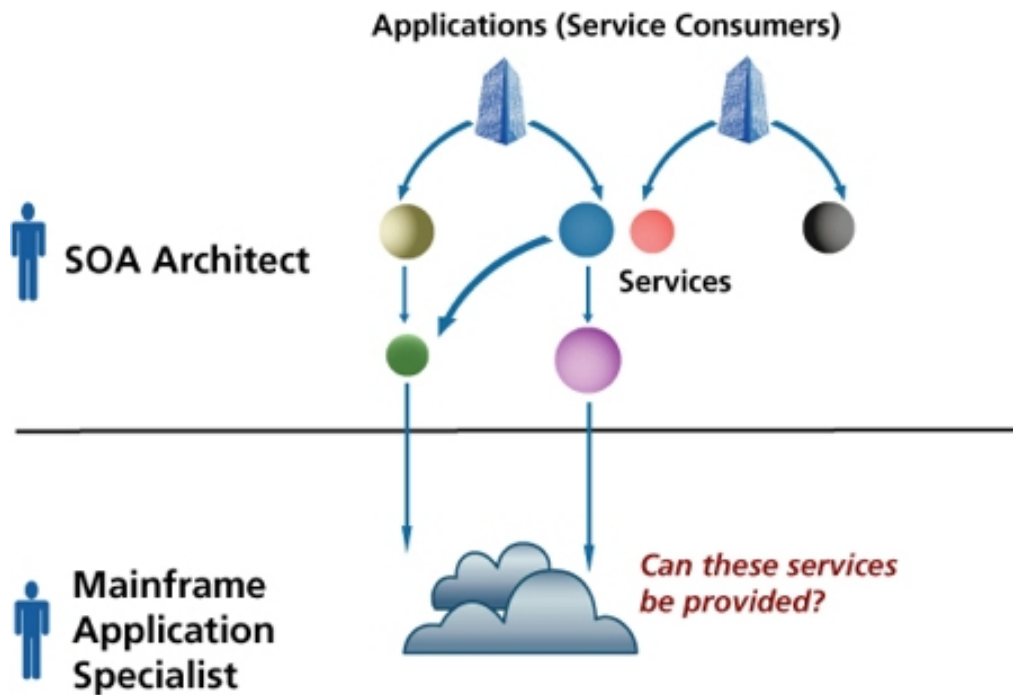


Figure 2: With the bottom-up method, the service definition is performed jointly by SOA architects and analysts and mainframe application specialists. Services candidates directly derived from legacy programs are a primary source of input.

Because either of these approaches can impose significant challenges, it can be preferable to use an alternative strategy known as meet-in-the-middle. In this approach the service modeling team and the mainframe application experts work together to identify potential services that are both useful and feasible, given the existing legacy constraints. Before solidifying a particular design, the architect would ask: "Can such a service be provided?" to which the legacy expert may respond "yes" or "no," or offer alternatives.

The mainframe expert is typically tasked with the responsibility of discovering if and where in the legacy application potential service operations may reside. This may be a relatively straight-forward process when working with small or medium-sized applications, but it becomes more difficult with large-scale legacy environments that can consist of hundreds or even thousands of programs, tens or hundreds of tables and files, and numerous screens, reports and external interfaces.

Accessing Legacy Application Layers

While the concept of a service may have not been used in the days of the mainframe, there were other concepts that closely resembled it. One familiar notion is that of *application layering*. This refers to an architectural style of designing and building the mainframe application in clear-cut layers, starting with screen-processing programs at the top and ending with data access programs at the bottom. There are many similarities with SOA, except that there was no intention to utilize the bottom layer as a real collection of “enterprise services” that could be accessed from outside of the application boundary or could be orchestrated (or choreographed) to build new functionality on demand.

Given their synergy with SOA, well-layered legacy applications are more suitable for harvesting potential services. One has to simply reach to the bottom layers and identify them. These bottom layers may consist of data access programs or other programs that use data access programs to assemble more coarse-grained services. The data access programs would form a logical data access layer, while the programs that implement some fundamental business logic would establish a separate layer above the data access layer. If the mainframe application architecture looks like Figure 3 below, these bottom layers can be easily tapped to identify and isolate useful services.

In this fortunate case the identification of proper services is easier, but it may get much more complicated when the layers of the application are not clearly delineated or when the application was built without any concept of layering.

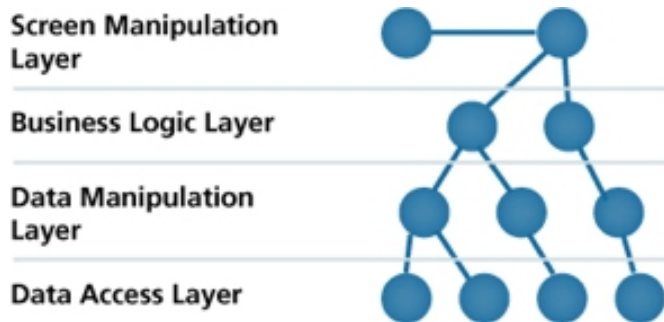


Figure 3: Legacy application layers closely resemble the multi-tiered nature of distributed solutions. That is why services are more easily extracted from these types of legacy programs.

Regardless of the approach being used, the challenge to identifying services can be further compounded by a lack of on-site expertise. Many IT shops suffer from a constant turnaround during which those most familiar with the inner mechanics of mainframe programs may retire or move on to new jobs. Over time, expert application knowledge is gradually lost or reduced to small islands of competence and lacking in the encyclopedic and all-encompassing expertise that would ideally be present.

The question for the legacy expert in the meet-in-the-middle approach is usually whether there is a particular program (or portion of a program or a set of related programs) in the legacy environment that performs a precise and well-defined manipulation of data, and which could be exposed as a service. Below are some possible ways to find out.

The "Walking Encyclopedia"

I was once demonstrating to a client the capabilities of a software tool to find answers to specific questions, like “What are the three conditions needed to change the status of an insurance claim?” While I proudly showed a method of scanning a 1000-program application to find the answer in less than three minutes, one of the developers in the audience exclaimed, “I already know the answer. The first condition is in the second paragraph of program PXB123P, the second condition is in the last paragraph of program PXC230Q, and the third actually appears in two places.”

If you have such a walking encyclopedia in your shop who can instantly answer any question related to the legacy application (and this person does not take vacations and does not plan to retire), then you may stop here and skip the rest of this article. In the absence of this type of miracle resource, you will need to explore other methods.

The Process View

Legacy application users are first and foremost familiar with the screens they access to carry out the tasks they need to perform. Such operations are intricately connected with the potential services that may lie within legacy programs. If the screens and operations on the screens are accurately described, they could easily become the starting point for finding services.

An effective method of capturing process logic is to express it via UML use case or activity diagrams. It helps if the identified use cases and activities are somehow mapped to the legacy artifacts that implement them. The mapping could take various forms, but it usually includes a cross reference, as in Figure 4.

Process	Legacy Artifact
Use case: Define Order	Programs: ORD030P, ORD031P, ORD035P Screens: ORD030M, ORD031M
Activity: Add Item	Programs: ORD051P, ORD052P, ORD053P Screens: ORD050M, ORD051M
Activity: Compute Discounts	Programs: ORD062 Screens: ORD050M

Figure 4: A typical process-to-legacy artifact mapping table.

A call map as in Figure 5 is also needed, as it helps the analyst start with a particular program and find all other dependent programs required to implement its functionality.

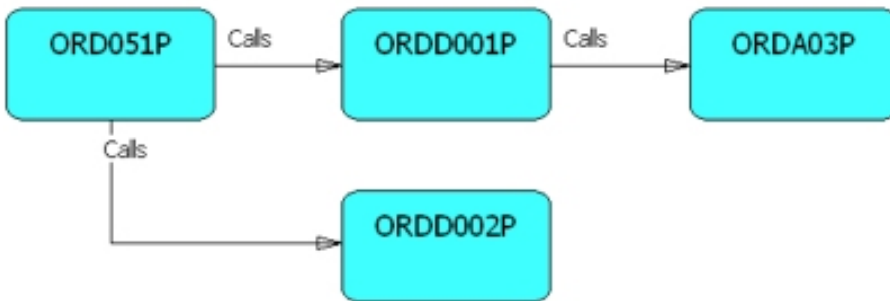


Figure 5: A mainframe call map is similar in concept to a UML sequence diagram in that it illustrates the invocation sequence of composed programs.

The identification of services may proceed with the SOA architect focusing on a particular activity that needs a particular service operation. For example, if the architect looks for a service operation that adds an item to an order, he or she will find in the table in Figure 4 that the corresponding mainframe function uses programs ORD051P, ORD052P, and ORD053P. If the first one is really performing the essential service of adding an item, the call map in Figure 5 would indicate that program ORD051P is calling two programs and the third is ORDA03P. While further detailed analysis would be required, such an analysis may quickly conclude that program ORDA03P is really the one needed.

At this point, a service operation could be defined. The input and output messages of this operation would roughly correspond to the inputs and outputs of ORDA03P. I say "roughly" because some fields in the communication area of ORDA03P may be superfluous and may not need to be part of the message.

The advantage of this technique is that it moves relatively quickly from the function of the service to the actual mainframe artifact.

The Data View

Another approach is to rely on a CRUD matrix of the application, as shown in Figure 6. The columns in this table correspond to the data stores used in the application while the rows correspond to the programs. Each cell at the intersection of a column and a row indicates the kind of operations that the program executes against the data store.

The operations are C(reate), R(ead), U(pdate) and D(elete).

Program/Table	Order	Payment	Order_Item	Item
ORDD01P	RU	-	R	R
ORDD02P	RU	U	R	
ORDA01P	RU	-	CUD	R

Figure 6: A classic CRUD matrix wherein create, read, update, and delete functions are identified.

Having a CRUD matrix available, the analyst could proceed as follows. From the service operation definition (for example, “add item to order”) he or she knows what operations need to be performed against what table. In our example, it must somehow involve the insertion of a row in the Order_Item table. The table above shows that the only program that inserts or creates a row in this table is ORDA01P; therefore, this is the one that must become the basis for the service. The program may be wrapped accordingly and the service messages may be inferred from the program input and output.

The Issue of Clustering

When the mainframe application specialist is looking for potential services, he or she will often discover programs that roughly correspond to service operations, and COMMAREAs of the programs that can represent the input and output messages used in these operations. A service, however, is a grouping of such operations and the open question is how this grouping should be determined.

Different methodologies lead to different ways of grouping operations. If a process view is taken, as explained above, it becomes easy to group the operations based on the fulfillment of some major transactions in the application. For example, a service may be defined for the processing of orders. This would group operations like “create order,” “fulfill order” and “cancel order.”

On the other hand, if a logical data view approach is taken, operations can be grouped based on a commonality of data. In this case, a service would be defined with more CRUD type operations.

The Issue of Granularity

As already established, multiple types of services can be derived from mainframe programs. However, not every potential service is a useful service. The overriding criteria relates to the business requirements of the SOA initiative. But another factor may be the level of granularity desired in the new architecture. One may build fine-grained services that perform detailed operations or coarse-grained services capable of carrying out more complete tasks.

There are various schools of thought regarding what constitutes the proper level of service granularity. While identifying services in a legacy application, the analyst may concentrate on a series of operations that perform a particular task, or on minute operations that manipulate the data. Below are some potential levels from the most coarse to the most fine or detailed.

- *Transaction Level* – The user of the application may have to operate on a number of screens in order to perform a particular task. More than one program is usually involved. For instance, to create an order, the user may start with a screen where he or she enters the customer information, then move to another on which a number of items are added iteratively, and finally enter the shipping information and commit the order. A new program may be written which would, in turn, call all the other programs involved in the proper order. This program would then become the basis for the “create order” operation.
- *Data Manipulation Program Level* – If a program already exists that is able to deal with the whole order at a time, it could obviously be used. If such a program does not exist, service operations may be defined at a more detailed level, which corresponds to existing programs. In this case, one may create service operations for “add item to order,” “update shipping information,” and “commit order.”

- *Data Access Program Level* – At an even lower level, services may be created to correspond to the basic operations against the data stores of the application. In this case, one would end up with programs like “add record to Order_Item table” or “update Order table.”
- *Paragraph Level* – It is possible that a particular operation is not implemented at all in a single purpose program, but is buried in the code of a program that has multiple functions. It may be necessary in such a case to slice out the desired operation, thus creating a new program specialized in just one function.

As you may have guessed, a combination of all of the above is sometimes needed.

The Issue of Negotiation

While the advent of SOA has been responsible for major technological advances, it has also imposed some culture shock upon IT environments that have proceeded with its adoption. For example, “mainframe people” have historically lived in an isolated world wherein they were responsible for running the heavy engines of the business. Now, with the introduction of service layers, these individuals are required to share this world and collaborate with “distributed technology people.”

The meet-in-the-middle approach forces these two communities to interact and negotiate. As with any negotiation that aims for a successful conclusion, there must be some give-and-take on both sides.

The fact that there is an existing legacy application that these groups need to collectively deal with can be both a blessing and a challenge. The good news is that there is a lot of functionality already there for the taking, but the downside is that this functionality does not usually match the most current business automation requirements.

But, more often than not, there is a compromise that can be reached; although the SOA architect may have to accept a less-than-perfect design, modern orchestration technologies can be used to combine legacy-derived services with custom developed services into sophisticated compositions that can fulfill most requirements. This means that even when the “perfect” legacy service is not available, a variety of design options will usually exist that can leverage whatever it has to offer.

The people responsible for the mainframe application may discover that they have more work than they expected. To start with, the existing mainframe application may abound in “SOA irregularities,” which are in no way mistakes from a mainframe architecture point of view, but particular configurations that make exposure of services difficult if not impossible. (One such irregularity is the mixing of screen manipulation and data access in the same program.) Besides such needed remediation, mainframe developers may also occasionally have to modify their programs to fulfill particular requests for the required services.

Software Support

It is worth noting that some of the activities described in this article may need to be supported by products and technologies. Many of them are either too complex or too time-consuming to be performed manually. Here is a list of common activities that can be supported by existing commercial software:

- UML extraction from legacy applications (with pointers to the legacy code).
- Automatic CRUD matrix generation.
- Automatic identification of application layers.
- Detection of legacy application irregularities.
- Slicing of specialized functions out of existing programs.

Leveraging modern technology helps reduce the risk when marrying legacy applications with service-oriented design approaches.

Conclusion

The reuse of legacy functionality holds an exciting promise of a more efficient approach to the definition of service-

friendly business operations. Under SOA, services must communicate with each other, but so, too, must the analysts and architects involved in the process of enabling SOA. The best approach requires a meeting of the minds. By achieving the right balance of top-down and bottom-up approaches to service-enablement, an organization can more quickly identify and define the appropriate services to address its specific business problems.

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL



[home](#) [past issues](#) [contributors](#) [official book site](#) [legal](#) [rss](#)

Copyright © 2006-2007 SOA Systems Inc. All Rights Reserved