Realtime publishers

The Essentials Series

SOA and Mainframe Applications

sponsored by

Attachmate®

by Dan Sullivan

Copyright Statement

Addressing Design and Life Cycle Challenges of Mainframe Applications in an SOA Environment

Mainframe applications, long the foundation of information management in large businesses and governments, are today joined in that central role by Service-Oriented Architecture (SOA) applications. These two types of applications complement each other and hold the potential to enhance and optimize their operations—but only if the unique design and life cycle challenges are understood and addressed. This series examines key aspects of SOA/mainframe integration projects and provides best practices for addressing common challenges facing application designers, developers, and project managers.

This, the first article in the series, examines the different architecture models and describes technical and management best practices for improving the development process as well as later stages of application life cycle management. Specifically, this article will examine:

- Two distinct application models
- How to bridge different application models, including technical and organizational management issues
- Best practices for managing mainframe applications in a SOA environment

Before examining how to bridge mainframe and SOA development efforts, it helps to understand key differences between mainframe and SOA application development.

Two Distinct Application Models

The mainframe and SOA models evolved at different times and under different constraints. Of course, all well-designed applications share common attributes such as reliability, robustness, and maintainability, but there are other characteristics that distinguish application models; it is these differences that must be addressed when integrating mainframe applications into an SOA environment.

Mainframe Applications

Mainframe applications are often designed for a well-defined set of high-volume transactions, such as authorizing a payment by credit card, posting a credit to an account, or updating inventory. These applications support relatively fixed business processes, so flexibility is often not as important as reliability and performance. This has led to common patterns in mainframe programming.

Mainframe applications tend to be monolithic. Consider the kinds of applications that might be needed in a credit processing business: one application may be an interactive payment system built using CICS, another application is a batch-oriented system for generating management reports, and another analyzes payment histories and flags accounts for credit review. Speed and reliability are top priorities for the interactive application, whereas the credit review application must be flexible enough to detect a variety of payment patterns indicative of high-risk accounts. In each case, an application is designed to follow a relatively fixed execution flow with full or near full control over data. These applications provide and consume services within a single application. There can certainly be coordination between applications, but they would more likely run in tandem than run with an integrated execution flow typically found in SOA applications.

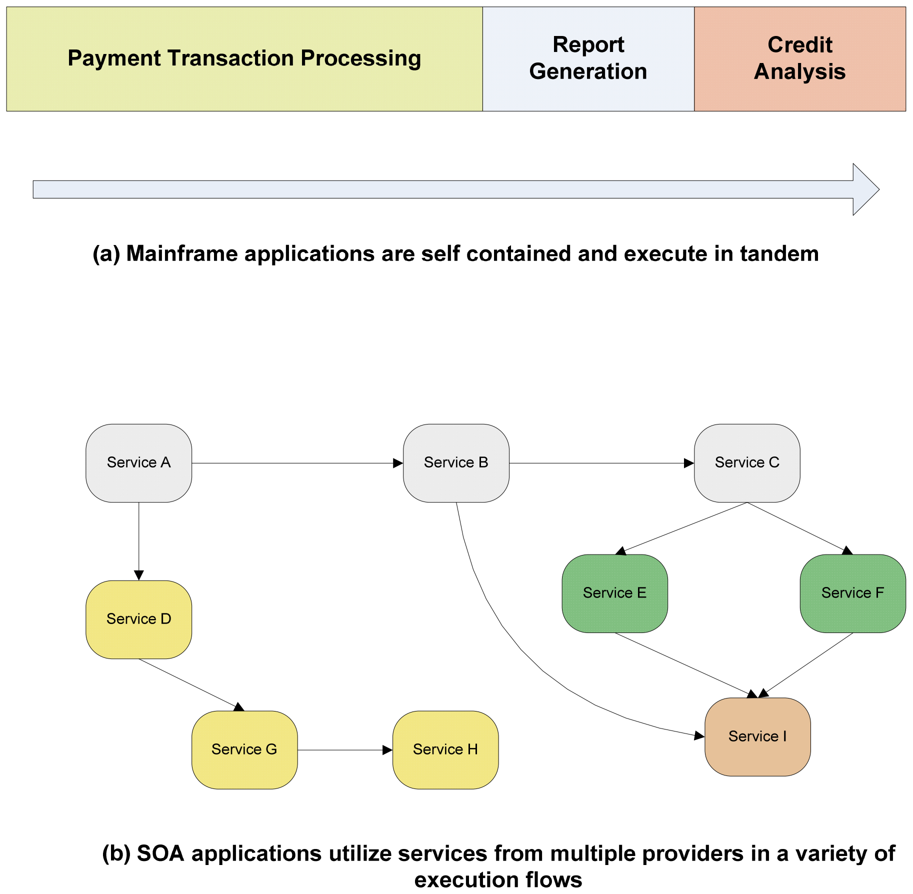
Payment Transaction Processing
Report Generation
Credit Analysis
(a) Mainframe applications are self contained and execute in tandem
Service A
Service B
Service C
Service D
Service E
Service F
Service G
Service H
Service I
(b) SOA applications utilize services from multiple providers in a variety of execution flows

Figure 1: Mainframe applications are designed for a broad business process, such as payment processing, and are relatively self-contained. SOA applications, however, are built from finer-grained services that are combined in a variety of ways to meet application requirements.

Mainframe applications are targeted to a single business process, so they can be optimized for that single operation. This, however, tends to come at the cost of reuse. It would be difficult, for example, to re-use the business logic of the reporting application in the risk analysis program without copying the code from one application to the other. That would be a relatively simple operation compared with having to maintain two versions of the business logic for the life of both systems. We can see how quickly this problem can grow as the number of applications expands and the amount of copied code increases. SOA applications, however, are designed to avoid this problem.

SOA Applications

SOA applications are designed for a distributed environment. Rather than design and build a single comprehensive application for a business process, SOA applications are built on a set of services that each carry out a specific task. For example, an SOA application in a credit card processing business might provide services for

- Retrieving a credit score from a third-party reporting agency
- Calculating a risk measure for an account
- Looking up the latest transactions against the account
- Generating an authorization code for a customer purchase

These services are made available to applications, which can use them as needed. There is no need to define and analyze all the possible ways a service may be used before it is deployed; other applications can call services as one would call a library function. In fact, developers have little control over how a service is called or the state of the larger application outside the service. Unlike mainframe applications, services and applications that call them are not closely coupled.

This lack of tight coupling between components is a key reason that SOA applications are reusable for distributed services. Data that needs to be shared between services is passed as parameters, and both the service provider and the service consumer agree to an implicit contract that defines what is needed to call a service and what the service will return. With this model, developers create business applications by orchestrating the use of existing services instead of building from scratch. The mainframe and SOA application models each have advantages, and combining the two can provide the best of both worlds—if done correctly.

Bridging Different Application Models

When leveraging mainframe applications in an SOA environment, it is important to remember that the goal is not to make a mainframe application into a distributed system or to make your Web services more like mainframe programs. The objective is to take advantage of the strengths of both modes.

Technical Bridges

Mainframe applications offer several advantages:

- They encompass a wide array of process logic, which when used in SOA applications, ensures the same level of data and process integrity found in mainframe applications. There is no need to duplicate code and maintain separate implementations of process logic when mainframe applications are used in SOA applications.
- They support high-volume transaction processing. These applications are designed to scale and are unlikely to become a processing bottleneck in an SOA environment.
- Mainframe applications provide services at a business-operation level, such as creating a checking account or a sales order. Compared with typical Web services, mainframe operations are course grained; but again, the goal is not to make mainframe applications more like SOA systems. These coarse-grained services include significant amounts of process logic and validation checks that are executed in the proper sequence and at the proper time.

Complementing these advantages is a different set of features that one finds in distributed applications:

- SOA applications integrate easily with other applications that support industry standards for Web services, including service transport protocols, such as HTTP and SMTP; XML messaging protocols such as XML-RPC and SOAP; the service description protocol WSDL and the service discovery protocol UDDI.
- Web services hide the implementation details of services provided. Service consumers and service providers agree on a set of parameters to pass to a service and the results returned by that service. There is no need for a service consumer to understand the internal state of a called service, making it easier to call these services than if there were side effects, which can sometimes occur with mainframe applications.
- Web services can provide an abstraction layer between mainframe and other systems more efficiently than implementing a service. For example, SOA tools and applications lend themselves to developing applications for mobile devices more readily than mainframe systems.

Just as there are different technical aspects of mainframe and SOA applications that lend themselves to different roles within an enterprise SOA environment, there are organizational differences around these application models.

Organizational Bridges

Successful mainframe and SOA integration initiatives address the need to build organizational bridges as well as technical bridges. One type of organizational bridge spans the need to manage different skill sets. One would not expect an orthopedic surgeon to perform heart surgery or a heart surgeon to perform joint replacement; similarly, we should not expect SOA and mainframe designers and developers to have each other's skills. Instead, project managers should divide development and maintenance tasks according to the mainframe or SOA skills involved.

For example, bringing a mainframe application to an SOA environment will require significant mainframe development skills. Developers must be able to analyze application code, determine dependencies, spot potential side effects of executing code, and assess the most appropriate level of a mainframe service. SOA developers have the skills to position a mainframe service provided by their colleagues at the appropriate level of the application stack, wrap the service to use standard protocols, and ensure that the service conforms to other SOA development standards.

Another area in need of an organizational bridge is life cycle maintenance. In fact, application development is challenging, but maintaining the mainframe/SOA system as an agile platform is even more difficult. Consider the fact that the safe use of mainframe components are typically not understood by SOA developers and designers.

Mainframe applications are designed around a different application model, and what is considered an acceptable or common practice in one type of application development may be unheard of or even eschewed in another. For example, code in one part of a mainframe application may set variables or change state information that is referenced in another part of the application. Not isolating the affects of code can make the code more difficult to understand but may be done for performance reasons. Also, mainframe applications are used for non-SOA functions that must be maintained and may require changes independent of the SOA programs. Mainframe developers are better able to spot these situations and maintain mainframe code without disrupting the integrity of the application.

The execution environment of mainframe and SOA systems are different and SOA developers may not understand the constraints on mainframe applications. A business process may require batch updates to secondary systems that occur in the middle of the night. Data returned from different mainframe applications may appear inconsistent because of time delays associated with batch processing. For example, a mortgage payment may be posted to the accounts payable system but not reflected in the account management system which provides payoff calculations until a batch operation is complete. An SOA that queried both systems may receive apparently inconsistent information when instead all the required steps in the business process have not been completed yet.

SOA developers are adept at integrating services and coordinating operations across services. Mainframe developers understand the logic of mainframe applications and the constraints on those programs. Allocating development and maintenance tasks according to skills is essential for efficient systems development and maintenance.

Summary: Best Practices

SOA systems and mainframe applications can together provide an agile IT infrastructure found in SOA environments with the performance and integrity long associated with mainframe applications. A few principles underlie best practices in this area:

- Leverage the strengths of both the mainframe and SOA application models. The models are complementary, not competing.
- Ensure developers with a proper mix of skills participate in integration efforts. No one person or type of designer has all the skills needed to span the mainframe and SOA worlds.
- Plan for a life cycle that incorporates the needs of both models. Mainframe and SOA applications evolved under different requirements and constraints but both can be successfully accommodated if the proper technical and organizational bridges are in place.

This combination of principles promotes a balanced approach to maximizing the benefits of combining mainframe and SOA systems without trying to force one into the other.