Realtime publishers

The Essentials Series

SOA and Mainframe Applications

sponsored by

by Dan Sullivan

Copyright Statement

© 2008 Realtimepublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimepublishers.com, Inc. (the “Materials”) and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtimepublishers.com, Inc or its web site sponsors. In no event shall Realtimepublishers.com, Inc. or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

Realtimepublishers.com and the Realtimepublishers logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners.

If you have any questions about these terms, or if you would like information about licensing materials from Realtimepublishers.com, please contact us via e-mail at info@realtimepublishers.com.

Managing Multiple Skill Sets to Support Mainframe/SOA Development and Maintenance

Merging the complementary technologies of mainframe applications and Service-Oriented Architecture (SOA) presents organizational as well as technical challenges. In this, the second in the series, we examine the organizational issues related to managing multiple skill sets needed to support the use of mainframe applications in SOA development efforts, including:

- The specific types of skills needed in a mainframe/SOA project
- Adapting different development models and styles to the hybrid initiative
- Best practices for realizing the benefits of both mainframe and SOA models

Developers and designers from mainframe and SOA backgrounds will need their domain-specific skills, but they will also need an understanding of their colleagues' different principles and practices. Project managers will similarly need to understand how to coordinate and combine the efforts of the technical staff.

Skills Sets for Integrating Mainframe Services into SOA

Presumably the goal of integrating mainframe services and SOA applications is to bring the functionality and scalability of mainframe applications to a diverse service-oriented development environment. It would be easy to make the mistaken assumption that the differences between mainframe and SOA development are analogous to difference in coding in COBOL and PL/I—there are different ways to define what the program should do but a developer would do roughly the same thing using either language. This is not the case. Mainframe and SOA development are specialized domains within information technology (IT) with their own models, terminology, and tools. The first article in this series examined, in detail, differences in the models; here, we will briefly discuss the different terminology and tools found in mainframe and SOA development efforts and examine the implications of those differences on project management and systems maintenance.

Similar Terms, Different Meanings

In spite of the fact that fundamental programming and design principles are the same across the IT spectrum, there are significant difference in how these are applied within the different objectives of and constraints on mainframe and SOA development. Consider two examples.

Although developers may share similar terminology, such as *services* and *transaction*, there are subtle but important distinctions between these terms in mainframe and SOA environments. A service to an SOA developer may be a relatively small application that provides a simple unit of work, such as calculating the state tax for an online order. To a mainframe developer, a service might be the entire order processing sequence of events that includes a rich set of application logic to check inventory levels, determine shipping method, update stocking information, and other operations required to maintain the integrity of business operations.

Also, mainframe developers are accustomed to working in high-volume, transaction-processing environments that take advantage of specialized services, such as Customer Information Control System (CICS), which has been widely adopted since its introduction decades ago. CICS is used in screen-oriented applications and typically depends on batch programs that run in the background. Unlike mainframes, where there may be a relatively limited number of distinct types of CICS transactions, SOA developers may think of a transaction as any sequence of service calls that are treated as single logical unit. The exact steps may vary depending on the particulars of the data being processed. The steps are treated as a transaction by virtue of being part of a single unit of work as defined by a programming framework, such as Enterprise Java Beans 3.0.

IT is fundamentally based on the principles of computer science and information management, but these broadly applicable disciplines leave a great deal of room for specialization. Both mainframe and SOA models are examples of specialized instances of IT. As the previous examples show, similar terminology can mean different things to different kinds of developers. In addition, developers have different tool sets.

Different Tool Sets

Mainframe developers work with tools for application development, data management, and security that are distinct to their IT environments:

- Transaction processing servers such as CICS that run on the z/OS and z/VSE operating systems (OSs) for managing high-volume transactions, such as updating customer account data
- Mainframe databases, such as VSAM, IMS and DB2—VSAM is a flat file DB most commonly associated with CICS applications, IMS is a hierarchical database useful in manufacturing or other application where data assemblies are common, DB2 is a completely modern (SQL) relational database; though flat file and hierarchical databases predate relational databases, they are still used for some of their specific attributes like performance and scaleability, while DB2 is the popular choice for modern MF applications and is growing in its data share, so more mainframe developers can be expected to have relational database experience
- Mainframe access controls, such as system authorization facility (SAF) and remote access control facility (RACF), are used to control access to resources on mainframe systems; SAF provides a centralized point of control to mainframe resources; such centralized controls are not usually found in SOA environments
- Broad sets of application logic implemented to protect the integrity of business processes:
    - Complex business rules may be embedded within COBOL or other programs that are executed at pre-established points in a job
    - Mainframe applications are designed in such a way that necessary business logic always runs
    - Services are not provided that would allow a program to execute one part of a business process without executing required business logic as well

SOA developers use different kinds of tools and data management systems that, although there are fundamental similarities, are distinct enough that they are used and managed differently:

- Transaction management across multiple services using Java Transaction Services and higher-level constructs, such as Seam conversations.
- Extensive use of relational and XML data sources. Relational databases, such as Oracle, Sybase, SQL Server, and MySQL and are more commonly used in SOA systems. XML data stores are also growing in importance, especially with the increasing need to manage semi-structured data.
- Distributed access controls and other security controls based on the WS-Security family of protocols and standards.
- Data transformation and integration methods to restructure and reformat data from diverse sources into a single format suitable for processing in an SOA application.

Neither skill set is inherently more important or better than the other; projects that encompass mainframe and SOA models will need skills from both areas.

Different Design Skills Working Together

If it were just a matter of separating mainframe from SOA components so that they each did their respective operations without interfering with the other, then mainframe/SOA projects would be much easier to design and manage. Unfortunately, there is no hard and fast boundary that allows the two approaches to coexist without the need to adapt to some elements of the other. To realize the benefits of mainframe services in an SOA environment, we must understand differences in the way services are provided and consumed in each application type.

Two Approaches: Monolithic and Fine-Grained

Mainframe applications provide and consume their own services. The term "monolithic" is often used to aptly describe these systems. They do not usually make calls to services provided by other applications but perform the operations themselves. For example, if an interactive COBOL program running on a z/OS operating system needs to update a customer's address in a database, it will use code within itself to perform the operation. That is not the case in a typical SOA application.

A customer self service Web application may have an "update address" feature that is composed of several components including: code to generate the interface using HTML and XML, server-side code to verify the structural integrity of the data sent from the client device to prevent SQL injection and related attacks, and finally, a call to a Java or .NET service that executes a SQL statement to update the relational database. In addition, not only are the components written in different languages and use different protocols but also they may all run on different client devices and servers. These separate services function together, though, because providers and consumers use implicit contracts about what each service expects as input and what it provides as output. This separation of functions impacts the way applications are written.

In the case of mainframe applications in which all relevant events occur within the same logical piece of code, procedural side effects are acceptable. A loan payment processing application on a mainframe might deduct a principal payment from an amount-due field in an IMS database and then increment another field that stores the number of payments made to date. A more finely grained set of services, like those found in SOA applications, typically perform smaller units of work, such as deducting the principal payment from an amount due. They do not make assumptions about sequences of events or try to maintain information about the global state of an application. For example, rather than look to a database field for the number of payments made, an SOA application may find querying the database for the number of payment records to be a more consistently reliable method of retrieving the information. The underlying assumption is that the information in the database may be entered and updated in several ways under a variety of circumstances. The contrary mainframe assumption is that changes to the database are made through the same program that does the querying and therefore all possible ways of making changes are available to developers. These differing assumptions are neither categorically correct nor categorically wrong; they both have uses.

Choosing Between Monolithic and Fine-Grained Services

Business operations supported by mainframes often do not easily decompose into fine-grained services. These applications carry out complex operations with multiple steps; as a result, they contain a great deal of business logic. Should these business operations be re-implemented in a more modular, SOA approach to ease their integration? Although this approach is appealing in theory, the practical considerations of actually restructuring and redesigning make this option cost prohibitive. Consider some of the challenges:

- The staff with the best understanding of the business logic embedded in mainframe applications is mainframe developers, but it is SOA developers who would have to implement finer-grained services.
- The original mainframe application will still be needed for the high-volume transaction processing it was originally designed for, thus leaving two sets of business logic that must be maintained and potentially doubling the maintenance costs of providing the service.
- There is the potential for the two sets of business rules to change over time, ultimately differing from each other. Inconsistencies can jeopardize the integrity of data and necessitate detailed analysis of audit trails to understand discrepancies in the data.
- Who will ultimately have ownership and maintain the business services provided by both sets of services?

The software maintenance life cycle is replete with management challenges in the best of circumstances. Trying to force a mainframe service into a set of finer-grained SOA services is like trying to force the proverbial square peg into the round hole—with enough force, it can be done, but the results will probably not meet expectations.

Ultimately, the effectiveness of using both mainframe and SOA services is largely influenced by management decisions about staff, development skills, and software maintenance. Fortunately, the practice of combining mainframe and SOA services has matured to the point where several management best practices have emerged.

Best Practices for Managing Mainframe Services in an SOA Environment

Both mainframe services and conventional SOA-designed services can contribute to the success of an SOA project if several principles are followed. First, clearly assign responsibility and ownership for services based on skills. Project management should assign ownership of controlling the business context of a service to those that have the skill sets appropriate to the applications from which the services are derived.

As noted earlier, mainframe programmers and SOA developers have complementary skills sets. They use different sets of tools and design applications in different ways. Both have created practices that reflect the design goals and the constraints of the systems they develop. Mainframe programmers may have detailed knowledge about business process as well as an in-depth understanding of how mainframe code implements that business process. SOA developers probably do not have the same level of experience with macro-level business processes. These programmers are more likely to be familiar with protocols and design patterns used in distributed systems. Services that require complex business logic and high-volume transaction processing are best assigned to mainframe developers. Specialized, fine-grained services that require familiarity with Web protocols, distributed security, and commonly used application stacks are best assigned to SOA programmers.

Second, weigh the need for granularity of services with the need to leverage business logic embedded in mainframe programs. If a mainframe application can provide a service that encompasses two or more fine-grained services, one may be tempted to re-implement the service. Before taking on the responsibility of maintaining additional software, consider how the mainframe service can be used to provide multiple services. For example, rather than call the three separate mainframe services, one after each of three fine-grained SOA services, call the mainframe service once after all three fine-grained services have executed. The sequence of events in an SOA process is not fixed. (In fact, the dance inspired term “choreography” is often used to describe the art of sequencing events in SOA).

Third, manage mainframe and mid-tier development teams in ways that support coordinated development. IT is not monolithic and IT professionals are not homogenous. Bringing mainframe services into an SOA application is more like managing a relay race team made of peers rather than a hierarchical organization in which some answer to the demands of others. Some parts of the overall design will be dictated by the best practices in SOA and in other cases it will be dominated by the practical constraints of existing mainframe applications.

Managing a mainframe/SOA development initiative is a challenge. There are multiple skill sets and different ways of designing and maintaining software that must be brought together, but when properly organized, they will complement, not compete with, each other.