

Realtime
publishers

The Essentials Series

SOA and Mainframe Applications

sponsored by



by Dan Sullivan

Article 1: Addressing Design and Life Cycle Challenges of Mainframe Applications in an SOA Environment.....	1
Two Distinct Application Models.....	1
Mainframe Applications	1
SOA Applications	3
Bridging Different Application Models.....	4
Technical Bridges	4
Organizational Bridges	5
Summary: Best Practices	6
Article 2: Managing Multiple Skill Sets to Support Mainframe/SOA Development and Maintenance.....	7
Skills Sets for Integrating Mainframe Services into SOA	7
Similar Terms, Different Meanings	7
Different Tool Sets.....	8
Different Design Skills Working Together.....	9
Two Approaches: Monolithic and Fine-Grained	9
Choosing Between Monolithic and Fine-Grained Services.....	10
Best Practices for Managing Mainframe Services in an SOA Environment	11
Article 3: Optimizing Mainframe Process Logic and Data Management Services in an SOA Environment.....	12
Access Points to Mainframe Services.....	12
Access Levels: Associated Attributes	14
Screen-Level Interfaces	15
Bridge-Level Interfaces	15
Transaction-Level Interfaces	15
Data Access Levels	16
Best Practices for Optimizing Process Logic and Data Access.....	16
Summary	17

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.

Article 1: 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.

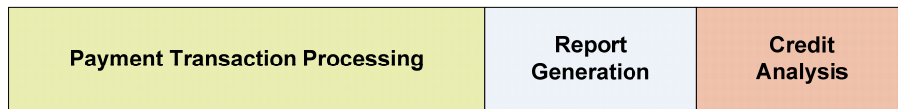
Two Distinct Application Models

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.

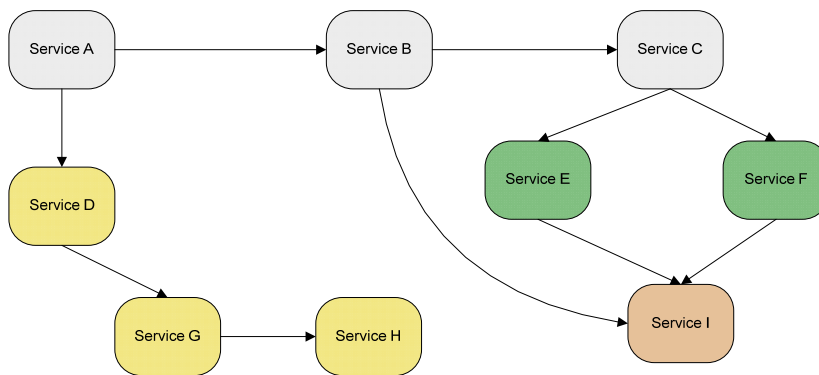
Mainframe Applications

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.



(a) Mainframe applications are self contained and execute in tandem



(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.

Article 2: 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 scalability, 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.

Article 3: Optimizing Mainframe Process Logic and Data Management Services in an SOA Environment

Once any questions about the need to provide mainframe services in a Service-Oriented Architecture (SOA) environment have been put to rest, designers must turn their attention to choosing the best way to make it happen. Fortunately, there are several ways to integrate mainframe applications. Determining the best method for any project depends upon the needs for process logic and data management services. In this, the third and final article in the series, we turn our attention to design and implementation details and examine three topics:

- Different access points to mainframe services
- The attributes associated with each,
- Best practices for choosing among the different access points

Mainframe services can productively function with more conventionally built Web services. Both can be described with provider and consumer metaphors and both use implicit contracts about inputs and outputs. However, beyond the application programming interface (API), services on mainframes and services in SOA stacks are fundamentally different.

Access Points to Mainframe Services

Application designers working with mainframe services face challenges not seen in conventional SOA applications. The foremost challenge is that mainframe applications are not designed for easy access by other applications. As noted in the previous articles in this series, mainframe programs are monolithic. They are typically designed to execute complex business processes in their entirety. For example, in a financial services company, a single application may be responsible for creating mortgages, processing payments, calculating account status, and grouping mortgages for sale on the secondary market. Any one of these operations might be made up of several services in an SOA architecture, as depicted in Figure 1.

From the perspective of a designer trying to leverage these models, the SOA model provides many more points at which one can gain access to a function performed by the application. For example, another application might want to provide data on when and to whom a mortgage is sold; there is no need to perform all the functions of secondary market processing. In the SOA model, it is likely that a programmer can make a call to one of the services supporting secondary market processing to retrieve just that information and no more. Any service in the logical stack of services is equally accessible.

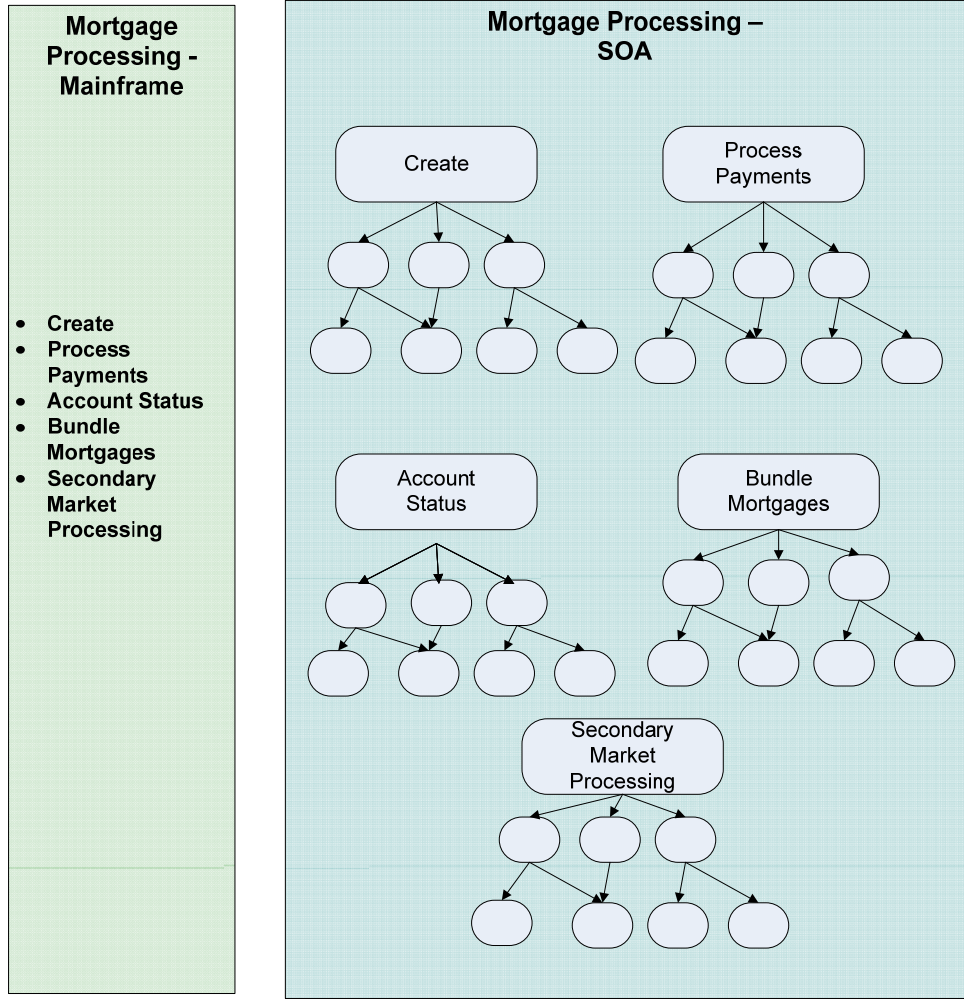


Figure 1: SOA applications present multiple points of entry unlike mainframe applications that tend toward monolithic designs.

SOA applications may have multiple layers of services with one service calling several others and each of those calls some number of other services. From a design perspective, there are no constraints on which services can call others. In some ways, mainframe applications can be thought of as having services, but they are more constrained than SOA services. In fact, it is probably more useful to think of them like the layers of an operating system (OS) in which each layer works directly with the layer above and below it. In this model, a layer consumes services provided by the lower layer and provides services to the upper layer. The service layers in mainframe applications are:

- Screen level
- Bridge level
- Transaction level
- Data access level

These are logical layers distinguished by the scope of services that are accessible in each level.

Screen-level interfaces provide the same level of functionality that one would find when using a screen-based application. When APIs are not available, this may be the only option for accessing mainframe services. This level provides access to the application logic with all of the intended User Interface flow control for governing the use of the logic.

The bridge level provides access directly to the CICS transaction server, which runs on the mainframe. For CICS applications, the majority of applications on the mainframe, the bridge-level technologies have access to all of the screen-level interfaces, but can also access application information from the BMS Maps. Bridge-level access takes advantage of IBM CICS Web Support and CICS 3270 bridge architectures by using messaging systems to communicate between the consumer application and the CICS transaction. Similar to the screen-level access, bridge-level access retains all the intended flow control built into the User Interface that governs the underlying logic.

Transaction-level interfaces access services made available through the CICS transaction level. At this level, there are no interface-specific details like those found at the screen level. At the bridge level, services are transaction operations that encapsulate the business logic of lower levels as well as transaction processing characteristics, such as isolated, atomic, durable, and consistent operations. This level of access provides a very fine-grain access to logic, but it does so by bypassing the governing flow control of any upper layer 'screens'.

The data access level is the lowest level of mainframe access. At this level, an application directly accesses data stores without the accompanying business logic found at higher levels. For example, at the data access level, an application could write a payment record to an account data file without updating a corresponding balance field.

From a purely information management perspective, no one of these levels is better than any other. There are, however, tradeoffs with regards to the amount of work assumed by the application calling a service at these various levels.

Access Levels: Associated Attributes

SOA application designers will have to weigh the inherited attributes associated to each of the access levels when determining the best method to deploy mainframe services. The following sections offer key points to consider.

Screen-Level Interfaces

Screen-level interfaces treat the user interface (UI) as a programmatic interface. This reduces the complexity of using and understanding the internal workings of the legacy application as it inherits the governing flow control built into the UI. While this alleviates the dangers associated with misuse of the underlying logic, it can put a burden on the consuming application to operate within the dictates of the screen interface's flow control. Also, this is typically not perceived as "programmer friendly" as conventional APIs. That disadvantage can easily be outweighed by the benefits of having all process logic executed with no additional cost to the consuming application. For example, a program that uses a screen interface to process a payment will realize all the quality checks and data integrity preserving operations that a human entering the same data would have. Historically, screen interfaces have been brittle; even the slightest change in a screen design could disrupt the way a consuming application worked. Tools designed for screen-level interfaces have improved, making this a safer choice than it may have been in the past.

Bridge-Level Interfaces

For CICS applications, bridge-level interfaces are a hybrid between screen-access and transactional attributes. Bridge-level access offers the ability to take advantage of the screen-layer's governing flow control of logic as previously discussed, but bridge-access can also see deeper into the underlying logic at a code layer. With this 'lower layer' ability, bridge-access is unaffected by changes in screen interfaces, avoiding what has historically been perceived as the brittle nature of screen interfaces. Even with advances in screen-level interface tools, when screen interfaces are too complex or may change unpredictably in ways that even newer tools cannot accommodate, bridge-level interfaces may be appropriate.

At the bridge level, consuming applications still have the benefit of executing process logic, which preserves the integrity of data and the application. Also, at this level, the mainframe service provides the quality of service one expects from a CICS native approach.

Transaction-Level Interfaces

As one descends the interface levels, one loses functionality. At the transaction level, one still has the benefits of transaction processing as inherited QoS of the CICS or IMS operating environments.

Transactions in CICS and IMS are units of work like a method in a java bean and should not be confused with transactional integrity.

At this level, transactions can be called as unique fine grain units of work, but this may be at a layer below any governing flow control. This implies that consumers of the CICS or IMS transactions know the context of the transaction's use within the CICS or IMS application as a whole – typically this is beyond the orchestration SOA skill-set and requires a 'COBOL' literate application developer be involved with any use (or re-use) of the transactions. Any needed flow control for use of transactions will have to be recreated in a composition layer of the SOA. Another potential, and possibly common, drawback is that applications that were not designed to be open to transaction-level interactions may not make their transactions callable.

Data Access Levels

Directly manipulating mainframe data at a data access level must be done with due consideration for a number of potential challenges. At this level, one is manipulating data fields directly without the benefit of integrity preserving application logic. An SOA application that functions at this level would have to re-implement the data and application integrity logic, but, if it could be done, SOA designers would still have to address problems with quality of service and scalability. These are mainframe strong points.

Even if business logic could be re-created within an SOA consumer application, there are potential coordination and synchronization problems if mainframe applications are manipulating the data at the same time. It is prudent to assume that mainframe applications are designed assuming that they have complete, monolithic control over their data. SOA applications manipulating mainframe data at the data access level must be carefully designed with full consideration of the assumed context in which mainframe applications function.

With these caveats in mind, there may be cases in which low-level data access is useful, such as when exporting data for use in a business intelligence application. Even in these cases, though, there may be situations in which expected data is derived from business logic implemented in higher levels of the access stack. With due consideration to the advantages and disadvantages of the various access levels, there are a number of best practices to keep in mind when choosing the appropriate access point.

Best Practices for Optimizing Process Logic and Data Access

The best practices for selecting an access level appropriate for a particular project are based on characteristics of the consuming SOA application services and the mainframe application that is providing services. First, one should choose an access level that supports multiple business requirements. If quality of service and transaction integrity is best controlled outside of the legacy application, a transaction level or lower access method is required. If the service must be relatively non-invasive because a mainframe developer may not be available to make changes, a bridge-level or screen access point may be most appropriate as the application's flow control is inherited.

Second, choose an access level that provides the required level of data and application integrity. If a consumer application only needs to read data and does not require any derived data, a data access level interface may be appropriate. If there are any updates to mainframe data, a transaction-level or higher interface should be used.

Third, consider the cost of service implementation over the life of the SOA application. A screen-level interface may be the fastest way to a functioning system, but if the screen interface changes frequently and substantially, perhaps a bridge-level interface would be more cost effective. Alternately, a transactional interface may be appealing, but if the loss of the governing legacy application flow control puts a burden on the mainframe application staff to assist in the use and re-use of the transactions in an SOA service, then a screen or bridge access method is preferable.

Finally, the over-arching principle in SOA/mainframe integration is comparable to the physicians' Hippocratic Oath: do no harm. When in doubt, choose a higher level access point so that process logic and data integrity measures are executed when a service is called.

Summary

Complex applications have been designed for mainframes and for SOA applications. Combining the two offers compelling operational benefits, but we must carefully consider the implementation details. Mainframe and SOA design models can work well together if design decisions are made with an understanding of the assumptions and constraints that guided the development of the mainframe applications and the needs of the SOA application.