Realtime publishers

The Essentials Series

SOA and Mainframe Applications

sponsored by

by Dan Sullivan

Copyright Statement

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.

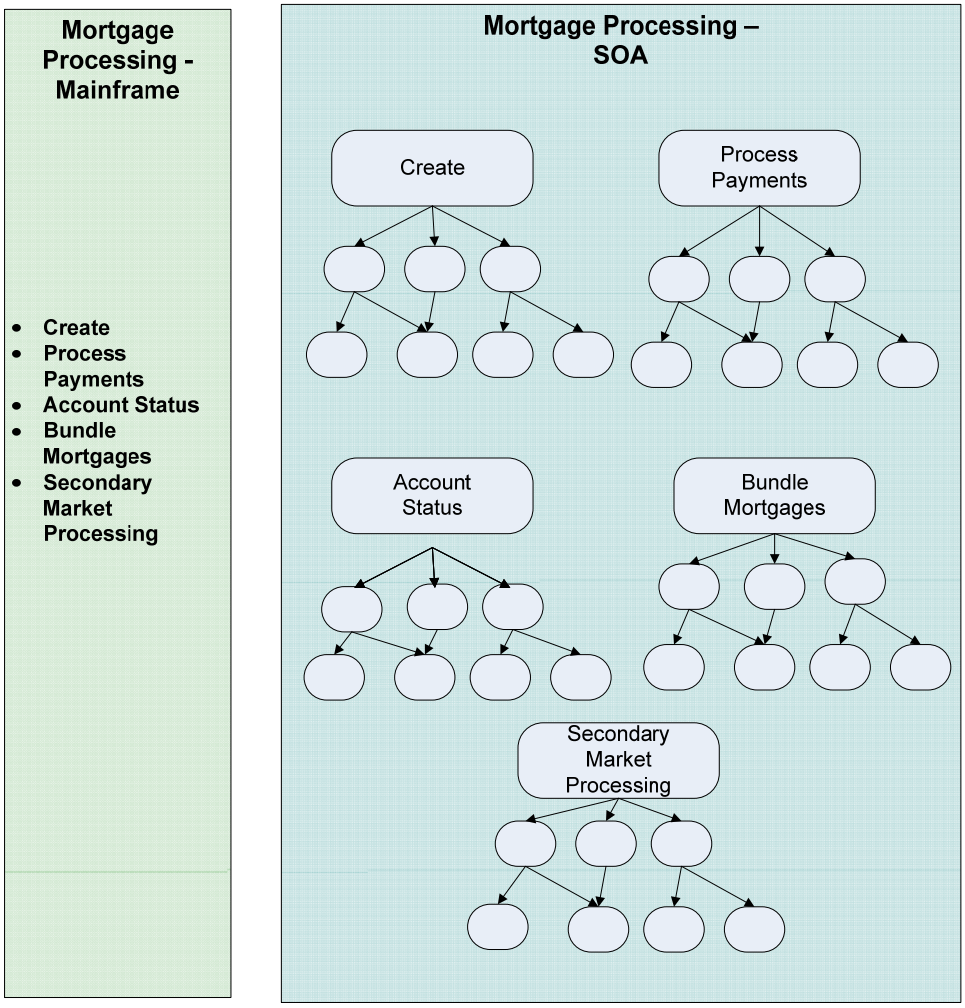

Figure 1: SOA applications present multiple points of entry unlike mainframe applications that tend toward monolithic designs.

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.