

Win in the flat world

Service-enabling Mainframe-based systems on Microsoft[®] Platform

- Sidharth S Ghag, Sachin Nayak, Vivek Kumar Shukla

This whitepaper has been developed as part of the Catalytic IT Legacy Modernization Solution

Abstract

Today's enterprises have applications that run on both Microsoft[®] Windows[™] as well as legacy systems. A large number of transactions are still processed by legacy systems such as IBM CICS (Customer Information Control System) and IMS (Information Management System). Of the Fortune 500 companies, 490 leverage CICS alone to process more than 30 billion transactions worth about \$1 trillion, each and every day^[a].

Due to the continued reliance on legacy computing, enterprises want to extend the life of mainframe systems and at the same time benefit from modern day technological advancements. Service-enabling legacy applications on the Microsoft .NET platform allows enterprises to realize this need. With the increasing adoption of .NET in several large enterprises, instances of .NET integration with mainframes are also on the rise.

This paper explores the challenges in mainframe integration and practical options of service-enabling mainframe systems using the Microsoft platform.



Overview

Mainframes have valuable information in terms of business logic and data, all locked in its closed environment. Despite being difficult to modify and unmanageable in the long-run, more than 70% of data and processing still reside on the mainframes. In today's distributed paradigm, organizations are faced with the huge task of unlocking this valuable information and providing it in a seamless manner to the other distributed IT systems in the enterprise IT landscape, thus enabling each system to benefit from the assets of the other. Service-enabling of mainframe assets can help enterprises address this issue.

Service-enabling is the process of exposing business logic and data embedded in the mainframe programs as well-defined, reusable services. This is achieved by designing services based on SOA principles and applying integration-based techniques discussed in this paper.

Why Service-enable?

Most organizations depend on legacy applications running on predominantly closed systems such as mainframes. Typically, IT departments in large corporations manage over 50,000 CICS code modules written in COBOL, Assembler, or PL/I^[a]. The oldest modules have been around for as many as 25 years, and still continue to process mission-critical data every day. Organizations devote a considerable chunk (as high as 75%) of their annual IT budgets towards maintaining and evolving this mainframe code^[a].

Enterprise technology has evolved rapidly over the past decade, and IT departments are required to extend the life of the legacy assets while at the same time deliver the benefits of modern day technologies such as Rapid Application Development (RAD), Smart Clients, AJAX, etc. Service-enabling legacy applications on the Microsoft .NET platform helps enterprises achieve this.

The costs associated with the migration of business data and customizable application code to platforms such as Windows, UNIX or Linux are considered to be significantly greater than the cost of their maintenance on legacy hardware. Also, the risks of such migration are high, and may include business disruption or even failure of the project. Hence, enterprises are seeking ways to service-enable and rejuvenate their mainframe resources.

Modernizing legacy systems using minimally invasive service-enabling techniques is a useful way to leverage existing investments on mainframe systems with minimum effort. With the increasing adoption of .NET by large enterprises, instances of .NET integration with mainframes are also increasing.

Challenges

Challenges faced in service-enabling mainframe systems are:

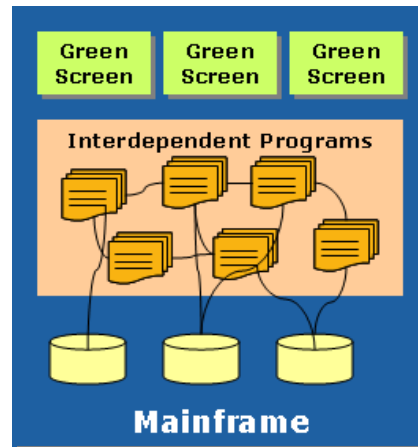


Figure 1: A typical mainframe system

Applications not tuned for integration

Most of the applications on mainframes are monolithic legacy implementation with limited layering (as in Figure 1), and have the business logic tightly coupled with the presentation interface and data access logic. This poor “separation of concerns” results in inadequate or non-existent interfaces which in turn restricts the integration of legacy modules with other applications.

It is difficult to service-enable some applications that contain tool-generated code because it is difficult to understand and re-factor¹. Also, they may contain code sections that are either redundant or are never executed.

Fine-grained legacy interfaces

Interfaces usually provided by mainframe legacy programs, be it at the screen or at the program level, tend to be fine-grained and require many method invocations for task completion. As against this, service-oriented interface designs are more coarse-grained at the document level and numerous types of data can be combined into a single request, thus requiring fewer method invocations for task completion. This mismatch tends to have performance implications due to the number of program calls a service might have to make. Sometimes, these calls are across a network and incur network latency overhead.

Hard to change and adapt

Complex interdependencies have resulted in legacy systems being unable to adapt to new requirements quickly. Organizations have ended up with duplicate code strewn across applications, requiring developers to carefully analyze the impact of changes, resulting in higher time-to-market for their services and offerings.

¹ Re-factoring is the process of changing a software system to improve its internal structure and re-usability without altering the external behavior of the program

Architectural Options addressing the challenges

Today's business environment is rapidly changing and has diverse, distributed and heterogeneous elements. To remain competitive, businesses must ensure quick adaptability automation and integration. Techniques which help businesses handle complex interoperability issues and produce flexible, compose-able solutions, within the constraints of costs and time are the need of the hour. Service-Oriented Architecture (SOA) is one such technique which promises to deliver not only all this, but much more!

The current trend in the software industry is to increase the levels of abstraction. Abstraction has established itself as an important tool to manage complexity. SOA attempts to abstract the integration concerns from the service provider and consumer agents. It hides the service implementation details and exposes only the contract (service interface) to the service consumer.

According to Forrester Research Inc ^[b],

Service Interface covers the functional design and documentation of what a service does.

Service Implementation covers the coding of the service's business logic within specific technologies and languages.

Discussed here are the various architectural implementation techniques available for organizations to address the above challenges. We focus on the integration approaches and the fitment for each of them in various usage scenarios for service-enabling. The techniques can be broadly categorized into the following three areas:

1. Presentation Integration Technique
2. Programmatic Integration Technique
3. Data Integration Technique

The above techniques employ deployment models which deploy the services interfaces either off-the-host or on-the-host^[c].

1. Presentation Integration Technique (also known as Screen Level Integration)

In monolithic² host applications that do not expose APIs and where the only interface with users is through the keyboard and screen, the Screen-based integration technique becomes the only viable choice to integrate such applications. Screen-based integration architecture enables developers to provide users with rich graphical user interfaces by using techniques such as "Screen Scraping". This approach involves extracting data from specific locations on the host application's terminal display for use in the Windows applications. Though this is a workable approach, it is considerably difficult to make the resulting applications robust and scalable.

As mainframe developers realized the benefits of separating presentation logic from business logic and data, they built applications that separated business logic interface components from user interface components. Consequently, presentation integration may not be necessary with newer host applications.

² the architecture where business processes and rules are embedded along with presentation logic

For years, because of the unreliability of direct database access (detailed later under the Data Integration technique), screen scraping was the accepted method of accessing mainframe logic from Windows or ASP applications. Screen scraping is necessary in some cases even today.

The level of runtime performance provided by a screen-based approach is simply not enough especially when dealing with enormous number of transactions, characteristically handled by large mission-critical mainframe applications. Despite this, sometimes this is the only option and has been successfully used for web-enabling a number of legacy applications.

Advantages:

- Non-invasive nature of creating web services
- Quick approach to service-enable the mainframe applications
- Connecting all host applications with one familiar programming interface results in lower training and administration costs

Disadvantages:

- Results in inflexible designs due to the tight coupling with screens
- Low runtime performance experienced for high volume transactions
- System may not be scalable to meet requirements

Currently Microsoft does not have screen integration capabilities and relies on its partners such as Attachmate, ClientSoft, Farabi, NetManage, and WRQ to build it. But Microsoft's Host Integration Server 2006, which is slated to be released mid-2006, is planning to introduce new screen-scraping technologies.

Scenario of usage:

- ✓ Application has green screens that have well-defined screen co-ordinates, which clearly demarcate input and output areas.
- ✓ Modules are highly monolithic and cannot be re-factored.
- ✓ No clear COBOL interfaces (i.e. copy books, which define the input and output of business logic programs on the mainframe) are available.
- ✓ Projects facing tight timelines.
- ✓ For low cost options, where a quick and dirty approach to service-enable is required.

Impact on the Mainframe application code:

This being a non-intrusive approach would not require changes to the existing legacy application code. A middleware such as Neon systems Shadow RTE, which provides screen integration components, would have to be installed and configured on the mainframe.

Architecture Options-

Screen-Scraping with

- a. Service Interface on Mainframe
- b. Service Implementation on Mainframe

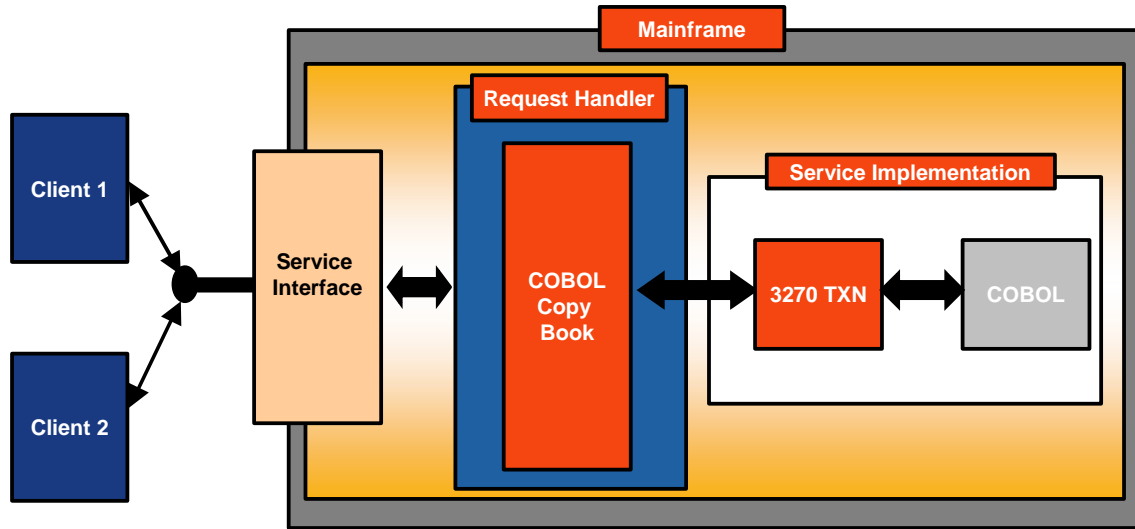


Figure 2: Presentation Integration

As shown in above diagram (Figure 2), both service interface and service implementation are on the mainframe. The entire request will be handled by a request handler, such as Shadow z/Presentation (which is installed on the mainframe used for screen scraping, where it captures all the events like transaction, and input/output fields). Services are implemented in the existing COBOL programs (without any modification) on the mainframe.

2. Programmatic Integration Technique

In this approach, the business logic residing on the mainframe will continue to reside there. Client applications communicate with the CICS system for the execution of the business logic programs. In this method, components will communicate with the mainframe via middleware such as MQSeries, or through communication channels such as TCP/IP via the CICS Transaction Gateway.

This integration architecture is suitable in situations where the presentation logic is clearly distinguished from the business logic.

This technique can be applied to a large number of host applications. Microsoft estimates that about 25% of CICS applications and 70% of AS/400 terminal applications are either pure program logic or have separate screen and program logic code. (All applications on IBM's Information Management System (IMS), a popular transactional and hierarchical database management system for the mainframe, have separate Screen and Program logic).

Advantages:

- Allows reuse of business logic
- Leverages investments made on mainframe
- Lower risks in migration
- Better performance and flexibility vis-à-vis presentation integration

Disadvantages:

- Poorly architected legacy code without proper layering may require invasive techniques (such as “wrapping”) to implement this approach
- Fine grained interfaces of the host programs would result in frequent trips to the host which would impact performance
- Continued investment on the maintenance of the back-end mainframe systems
- Data processing continues to be on mainframe

Scenario of Usage:

- ✓ This can be used in cases where applications on mainframe can be re-factored.
- ✓ Business logic programs have clean and well-defined interfaces.

Impact on the Mainframe application code:

Depending on the modularity of the legacy code, the impact on the legacy application code would have to be analyzed. Monolithic code where business logic is tightly coupled with the presentation interface and data access logic would need to be re-factored, to service-enable the mainframe. Re-factoring is done to separate the presentation logic from the business logic. By using dispatchers/listener type programs on the mainframe, business logic can be accessed directly. This enables better reuse.

Architecture Options:

- **Queue-based integration with both the service interface and service implementation placed on the mainframe system**

In this method the client applications place messages on queue-based middle-ware such as MQ Series. A program known as the listener program polls for these messages and passes them to the service interface. Similarly, responses are communicated back to client applications.

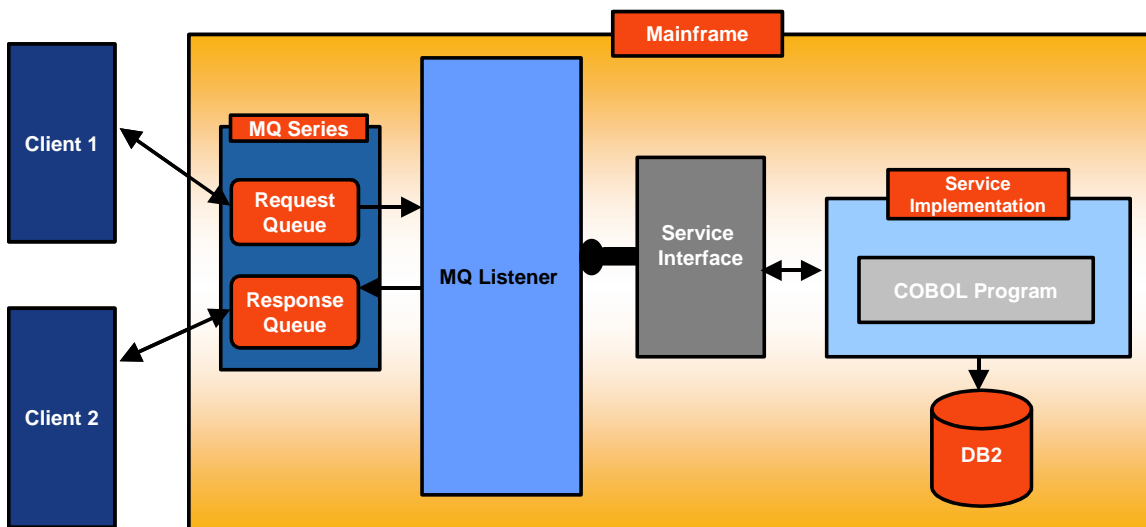


Figure 3: Queue-based Programmatic Integration with service interface on mainframe

As shown in Figure 3 above, both service interface and service implementation are on mainframe. The entire request is passed through MQ series via a listener. Services are implemented in existing COBOL programs on the mainframe.

- **Queue-based integration with service interface placed on the Windows environment and service implementation on the Mainframe**

In this option the service interfaces communicate with the service implementations by utilizing MQ Series.

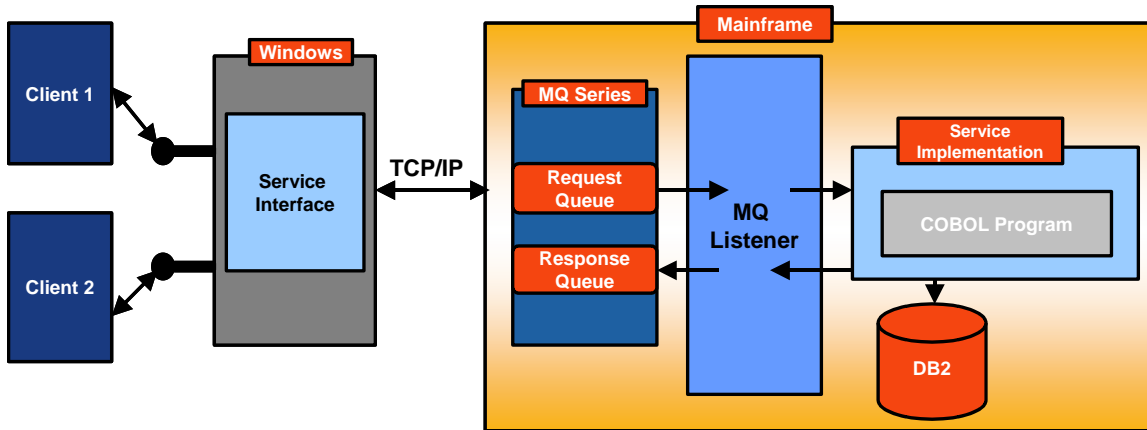


Figure 4: Queue-based programmatic integration with service interface on Windows

As shown in the above diagram (Figure 4), the service interface resides on the Windows machine and service implementation is placed on the mainframe. The entire request passes through MQ series via the MQ listener. Services are implemented as COBOL programs.

- **Middleware Integration with service interface on the Windows environment**

Middleware such as the Host Integration Server (HIS) enables architects to implement this option. HIS is a product from Microsoft, which provides various methods to abstract the mainframe. Features provided in this product include application integration and data integration. Application Integration is achieved by utilizing TCP/IP ports or SNA logical units to communicate with the mainframe.

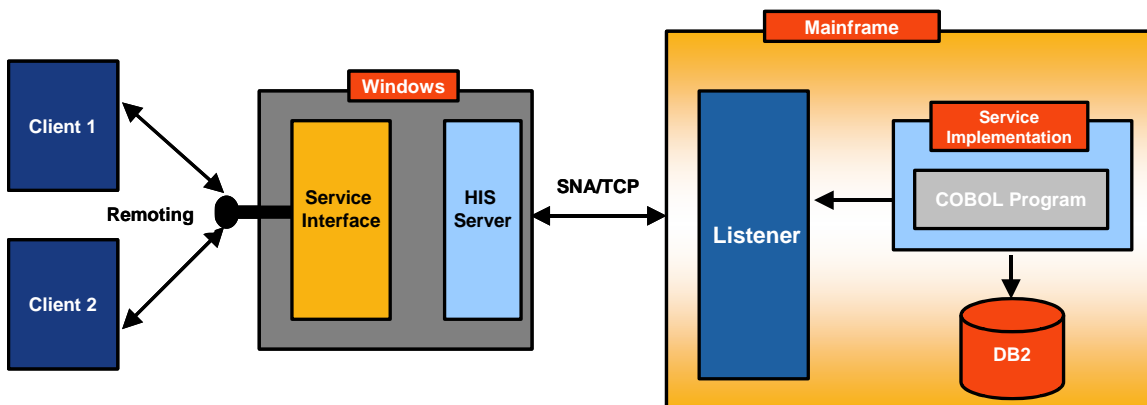


Figure 5: Programmatic Integration through Host Integration Server 2004

As shown in the above diagram (Figure 5), the service interface is placed on a Windows machine and service implementation on the mainframe. The entire request passes through middleware such as HIS. Services are implemented in COBOL programs.

- **Middleware Integration with both service interface and service implementation on the mainframe**

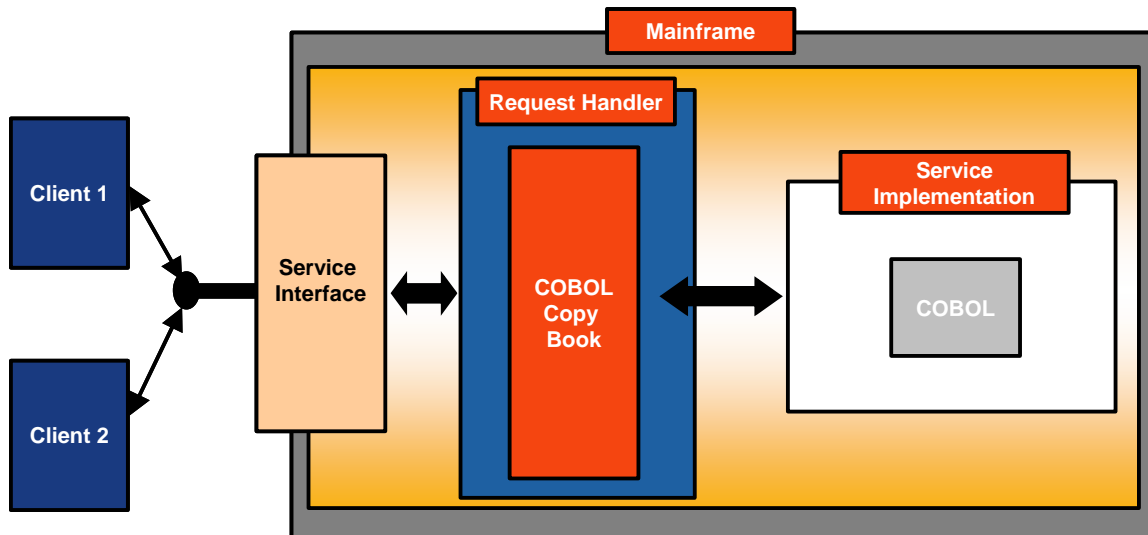


Figure 6: Programmatic Integration using middleware with services interfaces on the mainframe

As shown in above diagram (Figure 6), both service interface and service implementation are on the mainframe. The entire request is handled by request handlers such as Shadow z/Services from NEON systems, SOAP message handlers in CICS, etc. (handlers are installed on the mainframe, used for business level integration through "Comm Area" – a part of the linkage section of COBOL programs). Services are implemented in the existing COBOL programs.

3. Data Integration Technique

Direct database calls to raw mainframe data sounds like an ideal solution. Along with other advantages, this delivers relatively better response time, and allows the developer to repackage retrieved data in any format. However, by accessing the data directly, developers lose the real benefit of most mainframe applications viz. the business logic. In legacy mainframe systems, the business logic is tied to the user interface layer, so by going directly to the raw data, developers lose the ability to decipher the data. This technique essentially discards the business logic/functionality of the legacy system. Also, faced with a shortage of hard-drive storage space the original developers of mainframe systems stored most mainframe databases in various types of compressed files built on very confusing formats. Hence, this approach offers the least amount of reuse.

Most applications written for the mainframe have their data updated by batch jobs. So, modifying the data externally may result in data corruption when the external application updates the database.

Microsoft has various data access technologies in the form of OLE DB, ODBC, and ADO.NET data access providers which provide access to all kinds of relational data sources such as DB2 and non-relational data sources such as mainframe ISAM/VSAM and hierarchical databases such as IMS DB.

Advantages:

- Presentation and Business logic moved out of mainframe, saving MIPS usage
- Simple to implement due to the standard object models defined by the data providers
- Provides faster access to legacy data
- Provides ability for clients to develop business logic on a non-host environment
- Application framework requirements can be addressed

Disadvantages:

- Legacy programs containing business logic cannot be accessed
- Risk of data corruption is high due to the coupling between application and data
- Numerous formats for the compressed files may increase the time required for integration.

Scenario of usage:

- ✓ The burning need to reduce MIPS usage
- ✓ Developers building systems which do not need to access any business logic code and can directly integrate with the database. Application framework requirements such as event logging, instrumentation, etc. can access databases directly and can utilize this integration technique
- ✓ Data processing activity moving out from mainframe
- ✓ Move out business logic and presentation logic, but preserve data on the mainframe.

Impact on the Mainframe application code:

Since the application directly accesses the database on the mainframe, application code would not be touched.

Architecture Options

- **Data Level Integration with**
 - a. Service Interface on Windows
 - b. Service Implementation on Windows
 - c. Data on the mainframe

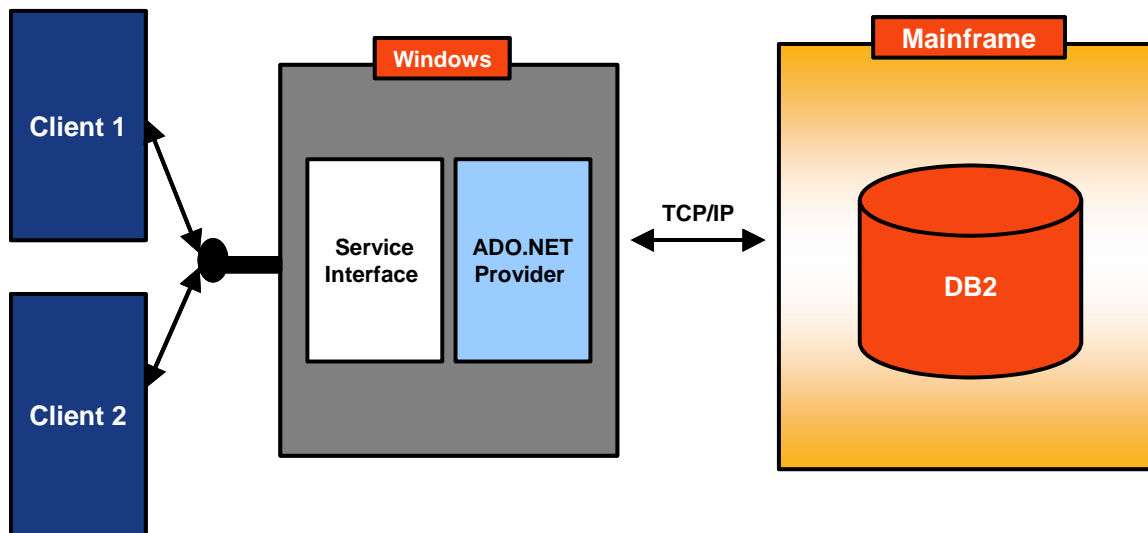


Figure 7: Data Level Integration

As shown in above diagram (Figure 7), both service interface and service implementation are on Windows and data resides on mainframe. Managed ADO.NET data providers for DB2 can be used to connect DB2 directly from Windows applications via Service Interfaces.

Benefits of Service-enabling the Mainframe

Service-enabling provides the following advantages:

- **Ready availability of information:** Inaccessible data on the mainframe is made easily available by adopting service orientation.
- **Reuse:** Service reuse promises to be the next higher level of reuse (higher granularity). For quite some time now, code reuse has been the most talked about form of reuse in software development. Unfortunately, it is hard to achieve due to language and platform incompatibility. Service reuse is comparatively easier to achieve. Service reuse at run-time is as easy as finding a service in the directory, and binding to it. The developer does not have to worry about compiler versions, platforms, and other incompatibilities that make code reuse difficult. Also, by sharing a single instance of a computing resource it aims to vastly simplify maintenance and ownership issues.
- **Productivity:** SOA provides the ability to quickly and easily create new and more complex services and applications using a combination of existing services, along with the ability to focus on the data to be shared rather than the implementation underneath. Additionally adopting .NET allows developers to benefit from the Microsoft Visual Studio®, a leader in the space of productivity, which provides the tools and utilities to enhance developer productivity.
- **Extend and revitalize the mainframe applications:** This allows enterprises to maximize the existing investments on the mainframe systems. The exposed services can be consumed across different delivery channels such as smart clients, web, PDAs etc allowing the enterprises to expand the reach of their services with minimum effort.
- **Technology Agnostic:** Enables enterprises to achieve a portfolio of applications that aggregate services existing on multiple platforms and different technologies, thus preserving existing investments while delivering new or extended functionality.

Benefits of Infosys Mainframe Integration Framework

- ✓ **Decoupling of Client and Mainframe application**
The business logic existing on the mainframe is separate from the logic of handling the service
- ✓ **Reduced complexity of integration**
The plumbing involved in integration with the mainframe is hidden from .NET-based clients. Transport specific adapters along with the serializers manage invocations of the mainframe code
- ✓ **Better scalability**
MIF is a stateless framework which allows developers to develop scalable SOA-based applications.
- ✓ **Service configurability provides the ability to replace services as needed**
New services can be easily plugged-in within the application due to the service configuration support provided by the framework. This allows developers to easily introduce new mainframe programs as Services on the .NET platform.

Conclusion

Most enterprises have applications and databases deployed on mainframes. These systems have proved to be very reliable and scalable platforms for handling large amounts of data and transactions. The huge amount of investment needed to re-architect and re-build these systems may not be justified for most enterprises. Only minor changes necessary to integrate them with the newer applications need be carried out.

Depending on the implementation technology of the mainframe system and the degree to which the Object Oriented principles have been employed during its design, the integration of mainframe systems may happen at the presentation, program or data levels.

Each of these approaches has both positives and negatives, and based on a situation the most appropriate approach should be adopted. Integration along with the principles of SOA, enables enterprises to service-enable the mainframe systems, and in the process leverage investments made on the mainframe as well as exploit technological advancements happening on the Microsoft Windows platform. Thus, benefiting from the best of both worlds!

References

- [a] <http://www2.sys-con.com/ITSG/virtualcd/Dotnet/archives/0107/raiford/index.html>
- [b] *Service Orientation: Key Application Design Principles*, Forrester Research, Inc., August 2003
- [c] "Legacy Enablement and Migration to SOA" by Dr Sriram Anand, Infosys Technologies, <https://infosys.webex.com/infosys/onstage/tool/record/viewrecording1.php?EventID=361197629>

Bibliography

- [1] "How to make the legacy assets more Business Process Management friendly?" by Binooj Purayath, Infosys Technologies Limited
- [2] "Mainframe Integration with .NET" by Hugh Raiford, ClientSoft published in .NET Developers Journal

About the Authors:

Sidharth S Ghag is a Technical Architect with the Microsoft Technology Centre at Infosys. He currently leads the Legacy Integration solution team and has extensive experience on various Microsoft Technologies. He has also been instrumental in enabling customers to service orient their legacy mainframe systems.

Sachin Nayak is a Technical Specialist at Infosys with over two years of experience in ASP.NET and legacy applications. His areas of focus are legacy integration and mainframe integration.

Vivek Kumar Shukla is a Technical Specialist at Infosys with over two and a half years of experience on Mainframe migration tools, Oracle and Microsoft technologies like .NET and SQL Server.

Microsoft® is the registered trademark of the Microsoft Corporation. Infosys Technologies acknowledges the proprietary rights in the trademarks and product names of other companies mentioned in this document which are being used without permission, and the publication of such trademarks is not authorized by, associated with or sponsored by the owners of such trademarks.

For more information, contact catalyticit@infosys.com

© 2007 Infosys Technologies Limited.

ALL RIGHTS RESERVED

Copyright in whole and in part of this document "**Service-enabling Mainframe-based systems on Microsoft® Platform**" belongs to Infosys Technologies Limited. This work may not be used, sold, transferred, adapted, abridged, copied or reproduced in whole or in part in any manner or form or in any media without the prior written consent of Infosys Technologies Limited.