

Integrating CICS Using SOA and Web Services

A HostBridge™ White Paper



Copyright Notice

Copyright © 2005 by HostBridge Technology. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without prior written permission. You have limited permission to make hardcopy or other reproductions of any machine-readable documentation for your own use, provided that each such reproduction shall carry this copyright notice. No other rights under copyright are granted without prior written permission. The document is not intended for production and is furnished "as is" without warranty of any kind. All warranties on this document are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Revision date: 6/6/2006

Trademarks

HostBridge is trademarked by HostBridge Technology.

Table of Contents

Service-Oriented Architectures	1
Web Services	2
Standards-based and Platform-independent	2
Reducing Interface Complexity and Increasing eBusiness Flexibility	3
CICS Applications	4
SOA Integration Models for CICS	5
Connector Model for Web Services	6
Gateway Model for Web Services	7
Dynamic Connectors versus Code Generators	8
SOA Deployment Options	9
Middle-tier SOA Boundary	10
Mainframe SOA Boundary (outside CICS)	10
Mainframe SOA Boundary (inside CICS)	10
CICS-based Process Automation (Micro-flows)	10
HostBridge and the SOA/Web Services Model	11
Enabling CICS Applications as Web Services	14
Conclusion	14

Integrating CICS Using SOA and Web Services

Service-Oriented Architectures (SOAs) and Web services combine to provide an opportunity for organizations to reduce the costs and complexities of application integration inside the firewall and open up new possibilities for legacy applications to participate in eBusiness. This white paper explains how you can use them to integrate mainframe CICS applications with other enterprise applications.

Web services are platform-independent interfaces that allow communication with other applications using standards-based Internet technologies, such as HTTP and XML. They provide an opportunity for organizations to reduce the costs and complexities of application integration inside the firewall and create new possibilities for legacy applications to participate in eBusiness. One problem with traditional integration techniques is the proliferation of point-to-point communication and data conversions that must change as new applications are integrated or data formats change. The problem gets complicated quickly when you add business partners, subsidiaries, mergers, or acquisitions into the integration mix. Web services simplify integration by reducing the number of APIs to one and the number of data formats to one.

Service-Oriented Architectures

Service-oriented Architecture (SOA) is a programming model that allows organizations to develop reusable software components, which communicate with each other across a common interface. An SOA is a client/server relationship where clients (requestors) invoke encapsulated business processes (services) using interface definitions published to service definitions repositories (brokers).

The interface is the most important part of a service. Whether the service performs a single unit, performs multiple units of work, or accesses one or several applications across the network does not matter; the requestor only needs to know the service definition: how to invoke the service and what to expect in response. This black box approach is essential to an SOA because it eliminates the need for integration developers to understand the underlying natures of each application they need to access and enables the basic request/response model.

SOAs are not dependent on any single technology for the development of interfaces. Previously, technologies such as CORBA and Distributed Computing Environment (DCE) provided a basis for SOAs, but could not gain broad enough vendor support to make an SOA practical throughout an enterprise. The recent rise in popularity of web services may be the catalyst that finally makes SOAs viable for organizations looking for ways to reduce integration costs while increasing flexibility.

Web Services

Web services are platform-independent interfaces that allow communication with other applications using standards-based Internet technologies, such as HTTP and XML.

Standards-based and Platform-independent

While applications have been able to communicate using Internet technologies for years, only recently have standards evolved to allow any web-based application to talk to any other web-based application. Web services depend upon three key open standards to allow them to communicate regardless of the hardware they run on, operating system they run under, or programming language used to produce them.

Table 1. Standards for web services

Standard	Description
SOAP	The Simple Object Access Protocol specifies a format for messages passed between web services.
WSDL	The Web Services Description Language describes web services so that other web services know how to access them, what to send as input, and what to expect as output.
UDDI	The Universal Description, Discovery, and Integration standard is a searchable registry that allows web services to publish WSDL documents so other web services can find them by name or category. UDDI also provides specification information concerning the input data formats, security models, protocols they use, and response data formats.

Web Services in a Nutshell

- Programs, not documents
- Remote programs invoked through a loosely-coupled messaging system
- Application interface based on Internet standards (XML and HTTP)
- Meant for application-to-application communication
- A family of technologies and specifications (SOAP, WSDL, and UDDI)
- Platform-independent
- Broad vendor support

Emerging standards for web services include:

- IBM's WSFL and Microsoft's WS-Routing for handling complex business process workflow.
- OASIS' SAML and Microsoft's WS-Security for handling authentication and message integrity.
- XAML and OASIS' BTP for handling transactions that span multiple enterprises and long lasting transactions to ensure that the outcomes of the transactions are reliable.

Underlying all of this is the use XML as the message format and a standard protocol such as HTTP as the transport. What makes these XML documents special is the consistent use of specific elements that allows interoperability among applications. The following examples illustrate how SOAP enables interoperability.

Standards-based components diminish the costs and skills required to integrate applications within an organization or between partners. Because there is widespread vendor support for web services, interoperability among vendor solutions will improve. However, the greatest benefits of standards-based web services for developers and administrators are reduced complexity and increased flexibility of integration architectures.

Reducing Interface Complexity and Increasing eBusiness Flexibility

The number of application interfaces and data formats that developers must learn and support affects the complexity and flexibility of integration. The more interfaces and formats the greater the costs to implement and maintain an integration framework. Web services provide a common interface to applications and XML provides a common data format — together, they simplify integration and lower integration costs.

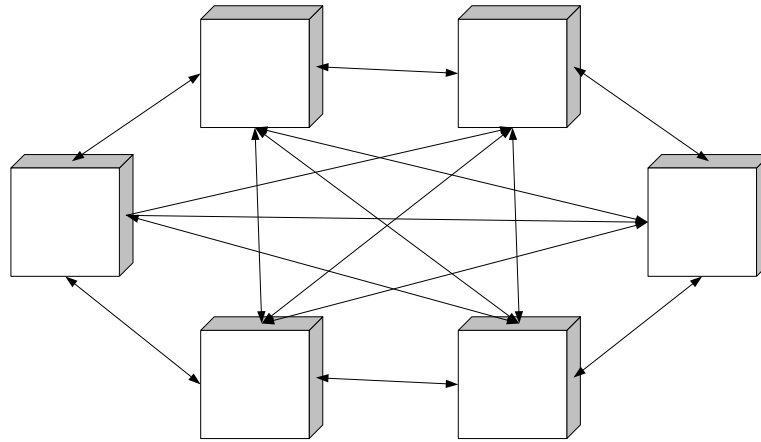


Figure 1. Traditional EAI frameworks (6 applications, 6 APIs, and 6 data formats) . Source: META Group

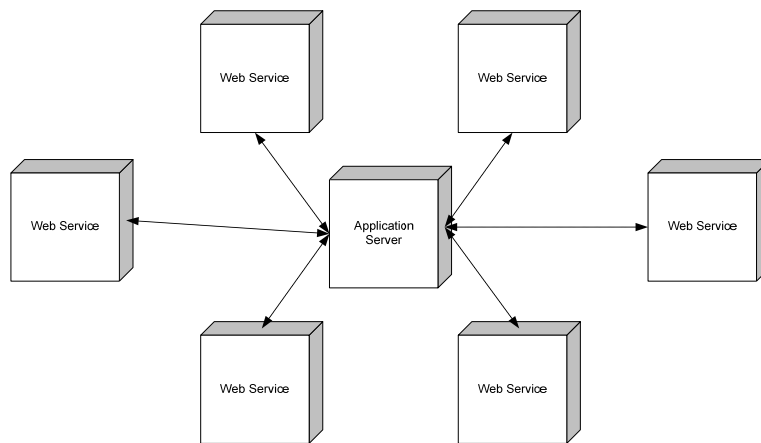


Figure 2. Web services integration frameworks (6 web services, 1 API, and 1 data format)

“With a new web services coating, existing systems, from CICS to client/server, will participate in new business scenarios. And with that participation codified in standard interfaces, the apps themselves can be modified or swapped out without bringing operations to a halt.”

Ted Schadler
Forrester Research, Inc.

Traditionally, developers create application programming interfaces (APIs) for each application in the integration project to allow inter-application communication.¹ Because the APIs for each application are different, companies integrating large numbers of applications develop complex integration frameworks that require specialized development skills and lead to increased integration costs. Increased complexity also makes integration architectures less flexible, so that changes in the network and business processes (following M&A, partnership changes, etc.) are slow and cumbersome.

¹ These interfaces — even interfaces built upon standardized interfaces such as CORBA, J2EE, or Microsoft’s DCOM — have platform-specific and vendor-specific components that keep them from being truly technology-neutral.

Reducing the number of APIs reduces complexity and increases flexibility. (See Figure 1 and Figure 2.) Web services based on Internet technologies are not specific to any platform and provide a single API for any application to use. They are loosely-coupled and can be invoked directly as traditional APIs or requests can be sent to a queuing system using where transactions can occur at specified dates or times.² Web services also provide greater flexibility when designing integration architectures. Because web services use a common interface, changes to the network or business processes do not affect the ability of individual applications to communicate with each other.

Another source of complexity is the proliferation of data formats used by each application. Reducing the number of data formats reduces the number of data transformations that take place as information passes through an integration framework. Figures 1 and 2 show how a common data format based on XML can reduce the complexity in an integration framework.

CICS Applications

IBM's CICS (Customer Information Control System) is a family of application servers that provides online transaction management and connectivity for legacy applications. Figure 3 shows a high-level taxonomy of CICS applications.

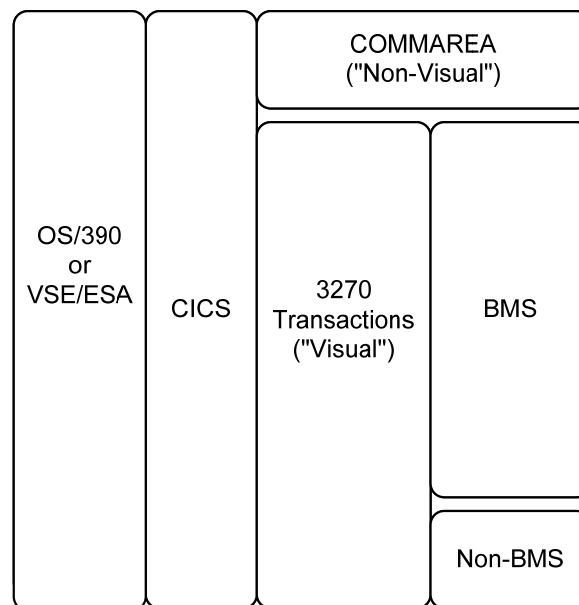


Figure 3. CICS application taxonomy

² Integration using traditional APIs is tightly-coupled in that each application accepts input and output using unique data formats and only recognizes direct invocation that expects to receive an immediate response.

CICS represents the broadest category of OS/390 applications. CICS transactions fall into two sub-categories: “visual” and “non-visual.” A “visual” transaction is one that expresses a presentation interface to an end-user at a terminal. You could also refer to a “visual” transaction as a “terminal-oriented” transaction. In contrast, “non-visual” transactions do **not** interact with an end-user. Instead, another program invokes these transactions. (This type of transaction is also referred to as “COMMAREA transaction” because the input/output parameters are passed to/from the transaction using an area of storage referred to as the “communication area,” or COMMAREA.)

The nature of CICS applications makes them complementary to SOAs and web services technologies. Non-visual COMMAREA applications take one request and return data to the requesting application in a single step. This maps well with the SOA/web services model because a single SOAP request would yield the required host data. Meanwhile, the majority of visual applications use a component of CICS called Basic Mapping Support (BMS). BMS essentially handles the presentation logic of the transaction and relieves the application developer from having to encode and decode 3270 terminal data streams. BMS expresses fields and data as name/value pairs, which can be converted to XML for consumption by web services.

Mainframes continue to be the most reliable and scalable platforms for handling large amounts of data and large numbers of transactions, so keeping your legacy applications on the mainframe is a good technical decision. Moreover, integrating your applications is usually cheaper than rewriting them. Now, by enabling your legacy applications as web services you can deliver integration projects faster and cheaper because your CICS groups and Internet groups use their existing knowledge to integrate the applications.

SOA Integration Models for CICS

“A myth has emerged in the industry that SOA and web services are the modern answer for the application integration problem. ... Even in the case of the composite SOA transactions, *additional integration technology is typically required* to reconcile the information model differences of the participating applications.”

Yefim Natis
Gartner. Inc.

In many cases, the costs of adding web services interfaces to mainframe applications are too high to justify. This means there must be an integration layer that accepts client requests and executes CICS applications on their behalf. There are two basic models for integrating CICS applications as web services, both of which include the use of adapters. The differences between these models depend upon where the web services exist, how they operate under the covers, and the types of applications you want to integrate. In this white paper, we will refer to these models as Connectors and Gateways.

- *Connectors* run on the mainframe and can use native interfaces that permit tight integration with the target application.
- *Gateways* run off the mainframe on middle tier servers and often use traditional methods, such as screen scraping.

In the purest sense, adapters might seem antithetical to the web services model because the target application is not itself acting as a web service. Nevertheless, when you face the daunting task of rewriting trillions of lines of code and millions of legacy COBOL/Assembler applications, the need for adapters becomes apparent. The question then becomes “what type of adapter do you want to use?”

"G3,500 firms have valuable business logic embedded in a hodgepodge of legacy systems -- from CICS to customized provisioning apps. Adding Web services interfaces to the mix makes those elements available for widespread reuse."

Laura Koetzle

The choice between using connectors or gateways often depends upon the types of applications you need to integrate. Integration vendors have traditionally provided facilities that allow you to use gateways to access CICS. These gateways commonly use techniques that capture legacy data based on row/column coordinates of the application screen (a.k.a., "screen scraping"). However, with the release of CICS Transaction Server, IBM began providing facilities that allow the use of connectors to access legacy applications, so you can choose between connector and gateway models based on your needs.

The recent availability of connectors that support the full range of CICS applications allows remote applications to invoke almost any CICS application as a web service. Because most shops have a mix of application types, companies should seek out this kind of connector to avoid multiple software licenses and additional training on how to integrate the different application types within your organization.

In the connector and gateway examples below, we focus on using web services with visual (terminal-oriented) applications because they are more difficult to integrate than non-visual (COMMAREA) applications.

Connector Model for Web Services

Connectors allow you to transform your legacy applications into web services without requiring the use of additional hardware, without changes to the legacy application, and without falling back upon brittle techniques like screen scraping. Compared to gateways, connectors yield better performance by running on the host and more reliable operation due to the elimination of the many layers data must pass through due to screen-scraping. (Figure 5 details these layers.) Connectors also provide enhanced access to application information such as state and error codes. This information is lost when data is sent to a terminal emulation technology such as a 3270 emulation client.

Figure 4 below shows the basic model for accessing legacy applications as web services through connector technologies.

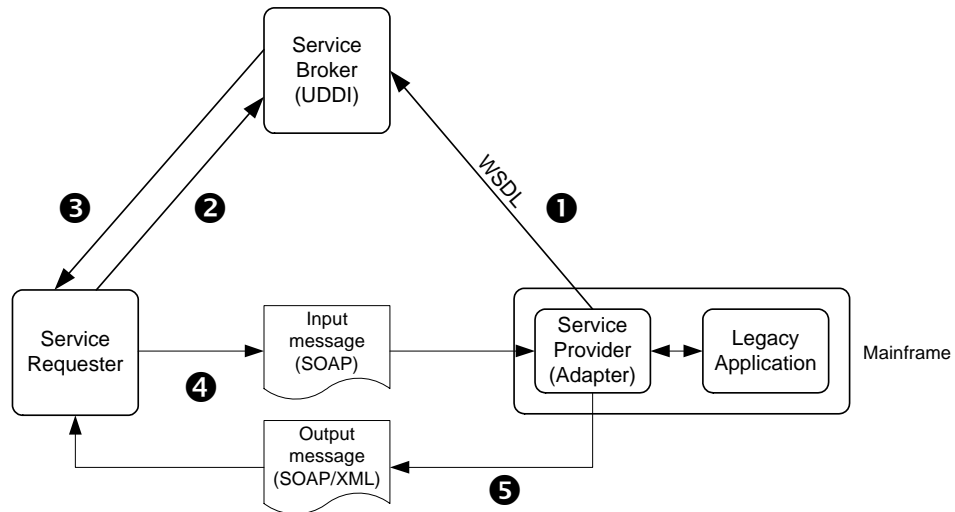


Figure 4. Connector model

The diagram in Figure 4 illustrates a web services architecture: a Provider that supplies the web service, a Requester that uses a web service, and a Broker that finds Providers for Requesters. In this model, the legacy application is the

Provider. The following steps represent how to find and use a web service connector. (Steps 1-3 are optional.)

1. The Provider uploads a WSDL specification to publish its web service with a Broker.
2. The Requester (usually a Java or .NET application) queries the Broker for a web service by name or category.
3. The Broker selects a Provider and returns the Provider information to the Requester.
4. The Requester uses the information from the Broker to format and send a SOAP message to the HostBridge.
5. The Provider returns a SOAP/XML response to the Requester with the legacy application data enclosed.

Gateway Model for Web Services

Unlike connectors, gateways typically run on a physical or logical middle tier. Where the gateway runs is important because there are so few options for accessing the host from the middle-tier servers, which means gateways usually involve some form of screen scraping. The solution is tightly coupled in that the integration is between the gateway and a specific application. Any changes to the application will break the integration.

When gateways communicate with terminal-oriented legacy applications they open a terminal session with the legacy application, send a request to the application, receive the terminal datastream, use HLLAPI to capture the screen data, process the screen data, convert the contents to XML, and ship the XML document to the requester. (A variation of the gateway model is to use FEPI on the mainframe instead of a middle-tier terminal emulation client. This simply moves the middle tier onto the mainframe.) The most common components of the gateway model appear in the Figure 5, below.

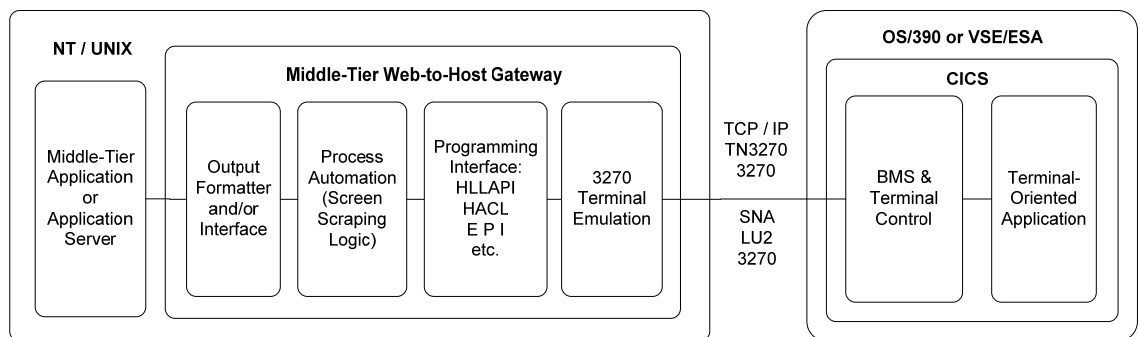


Figure 5. Typical middle-tier gateway architecture

Gateways allow you to get an application into the web services mix, but screen scraping creates performance bottlenecks and multiple points of failure between the legacy application and the web service. For this reason, gateways are best for short-term projects, either as a transition to using connectors or as a stopgap measure during application reengineering or platform migration.

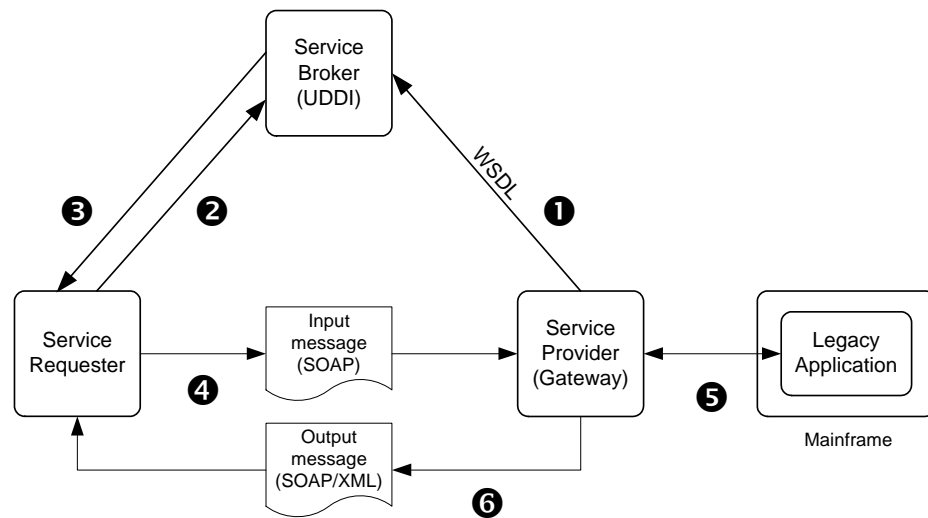


Figure 6. Gateway model

Figure 6 shows the basic model for accessing legacy applications indirectly using a gateway technology such as a screen scraper. Again, the diagram illustrates a web services architecture: a Provider that supplies the web service, a Requester that uses a web service, and a Broker that finds Providers for Requesters. The legacy application is not itself a web service, but is accessed by an off-host Provider. The following steps represent how to find and use a web service gateway.

1. The Provider uploads a WSDL specification to publish its web service with a Broker.
2. The Requester (usually a Java or .NET application) queries the Broker for a web service by name or category.
3. The Broker selects a Provider and returns the Provider information to the Requester.
4. The Requester uses the information from the Broker to format and send a SOAP message to the Provider.
5. The Provider starts an emulation session and conducts a series of transactions with the legacy application to collect the requested data.
6. The Provider converts the legacy datastream into an appropriate returns a SOAP/XML response and returns the response to the Requester with the legacy application data enclosed.

Dynamic Connectors versus Code Generators

As we have seen, connectors provide several advantages over gateways when it comes to web services integration. How those connectors operate can be just as important. Products that implement the connector model for web services usually fall into two camps: those that use code generation and those that are dynamic. Code generators have been around for some time and require developers to adhere to a structured process. In the case of CICS applications, something like the following takes place:

1. Download program source code, copybooks, or screen maps to a workstation running a proprietary tool.
2. Using a proprietary tool to generate web services “wrapper” code.
3. Upload generated code to the mainframe.
4. Compile, install, and test the generated code.
5. Repeat this process for each program and when a program changes.

While generated connectors can jumpstart the initial development process, they create several maintenance problems. First, the integration developer is generating code off the host and uploading the code to the mainframe. In most cases, this developer will know a lot about web services and XML, but will know little about the mainframe. Thus, changes to the mainframe application require coordination between the web developer, the application developer, and the CICS administrator. Second, generated connectors create “net new code” that must be managed. Changes either to the legacy application or to the web service will require repetition of the code generation process to keep the generated code in sync with the integrated application. Without proper management, you are likely to drown in a sea of generated code.

These issues led to the development of dynamic connectors. Dynamic connectors operate with little or no configuration and they automatically incorporate changes to legacy applications into the SOAP/XML output. In many cases, there is no configuration required, while some cases may require a single step process to specify web service information for each application. As a result, there is no generated code and there is a clear division of labor: the CICS administrator installs the connector and the web developer simply invokes the connector as a web service.

SOA Deployment Options

One of the first decisions facing enterprise architects is where to place the SOA boundary; that is, where the service provider processes the service request. Two factors influence this decision: the choice between Gateway and Connector models and the type of legacy applications on the mainframe.

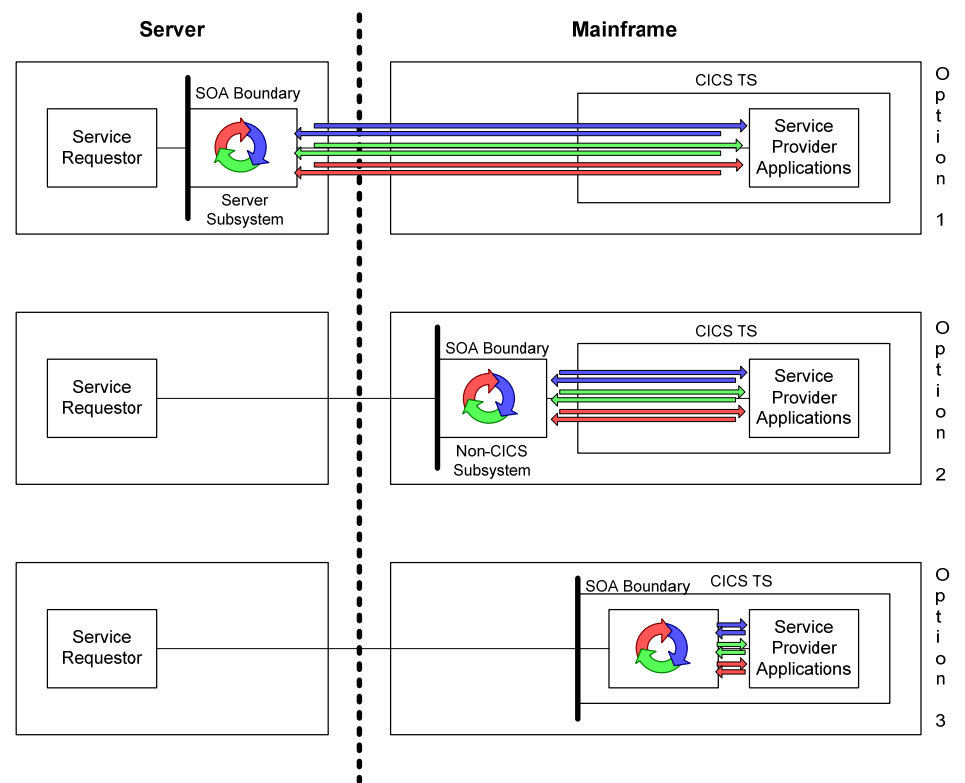


Figure 7. SOA boundary options

Middle-tier SOA Boundary

Organizations that choose the Gateway model for their SOA implementation typically place the SOA boundary on a middle-tier server alongside the gateway software. Most gateways include SOAP support to allow developers to include the screen interactions in an SOA. However, adding support for SOAs does not diminish the scalability, performance, and reliability issues associated with gateway screen scraping. This approach works equally well for any screen-based mainframe applications and data sources.

**Mainframe SOA Boundary
(outside CICS)**

Organizations that choose the Gateway model have the option to run the gateway on the mainframe outside of CICS. Likewise, organizations using the Connector model can move the SOA boundary outside of CICS. In the case of mainframe-based Gateways, moving the SOA boundary onto the mainframe can diminish some of the performance and scalability issues, but the reliability issues associated with screen scraping remain. For Gateways on the mainframe, the issues are the same for both CICS and non-CICS applications and data.

Connectors with SOA boundaries outside of CICS typically develop their own HTTP servers and often develop proprietary interfaces in an effort to overcome the 32k size limitation for moving data into and out of CICS over EXCI. This approach poses two problems for customers. First, regardless of the interface, moving data into and out of CICS for processing creates latency issues that vendors can avoid by moving the SOA boundary into CICS. Second, developing proprietary servers and interfaces rather than using the existing infrastructure that IBM provides as part of CICS Transaction Server means that customers must look to the integration vendor for support on infrastructure issues as well as integration issues.

**Mainframe SOA Boundary
(inside CICS)**

Organizations that choose the Connector model for their SOA implementations have the option to place the SOA boundary inside CICS on the mainframe. If the target applications are CICS applications, this approach eliminates the issues inherent in placing the SOA either on a middle tier or on the mainframe outside of CICS. A CICS-native SOA boundary allows the elimination of screen scraping for most terminal-oriented applications, which increases the reliability of the solution versus Gateway options. This option also reduces the latency introduced when vendors place the SOA boundary outside of CICS. Moreover, CICS-based connectors can eliminate vendor support issues by taking advantage of existing IBM technologies, such as CICS Web Support (CWS), SOAP for CICS, Link3270 Bridge, and the CICS Pipeline.

CICS-based Process Automation (Micro-flows)

The notion of “process automation” for CICS integration involves managing the flow of data between the integration software and the target applications or data sources. A common use for process automation involves navigating through multiple CICS transactions to obtain the data required by a distributed application. In a Server-Oriented Architecture, distributed applications can invoke the automated process using a service request.

In a scenario where the SOA boundary resides inside CICS on the mainframe, the service provider executes the micro-flows on the mainframe and eliminates the need for the middle-tier application to manage a complex flow of requests and responses. This scenario also allows customers to adopt a one-request/one-response model that reduces latency and simplifies architectures. (See Figure 8.)

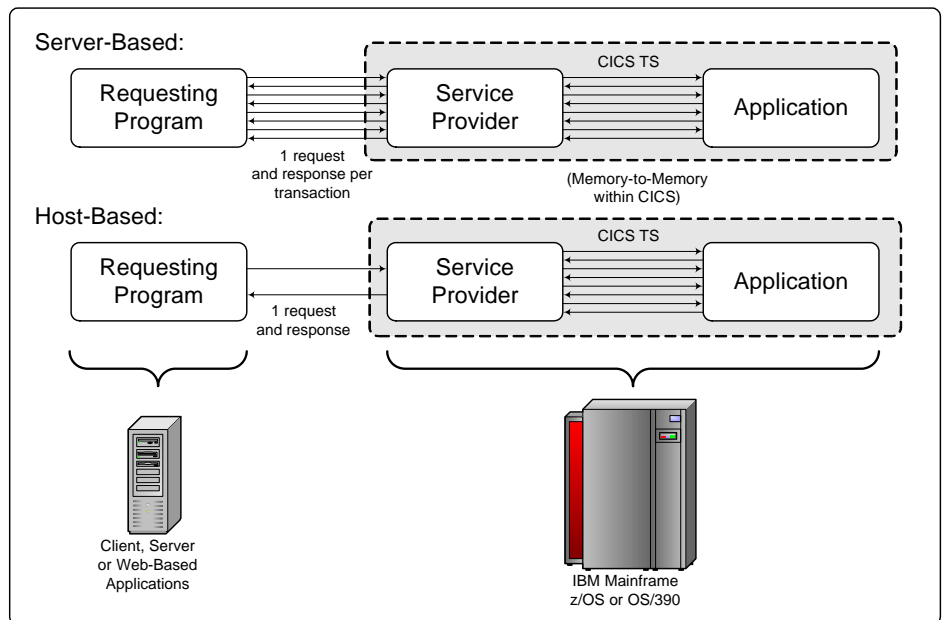


Figure 8. CICS-based process automation reduces latency; simplifies architectures

Corporate data rarely resides in a single application or data source, and integration projects often combine data from multiple sources into a single application interface or make them available through a single business process. CICS-based process automation allows developers to execute CICS transactions, query databases, and combine the returned data into a single XML document for use by requesting programs.

HostBridge and the SOA/Web Services Model

HostBridge is mainframe-based software that provides immediate access to your CICS resources as XML documents or web services in a Service-Oriented Architecture (SOA). The single largest headache a company faces when designing integration frameworks is dealing with legacy applications that contain decades of business processes and data. HostBridge preserves investments in CICS applications and lowers the risks and costs of integrating them with other applications throughout the enterprise.

HostBridge provides the scalability, reliability, and performance expected of mainframe solutions and combines it with uncommon simplicity so it easily fits into any XML or web services integration architecture. By using industry-standards like XML, SOAP, and Websphere MQ, HostBridge saves companies time and money needed to integrate disparate applications. HostBridge immediately extends the ROI of CICS resources and brings new returns through business integration.

The diagram in Figure 9 shows the various connectivity options when using HostBridge for CICS integration.

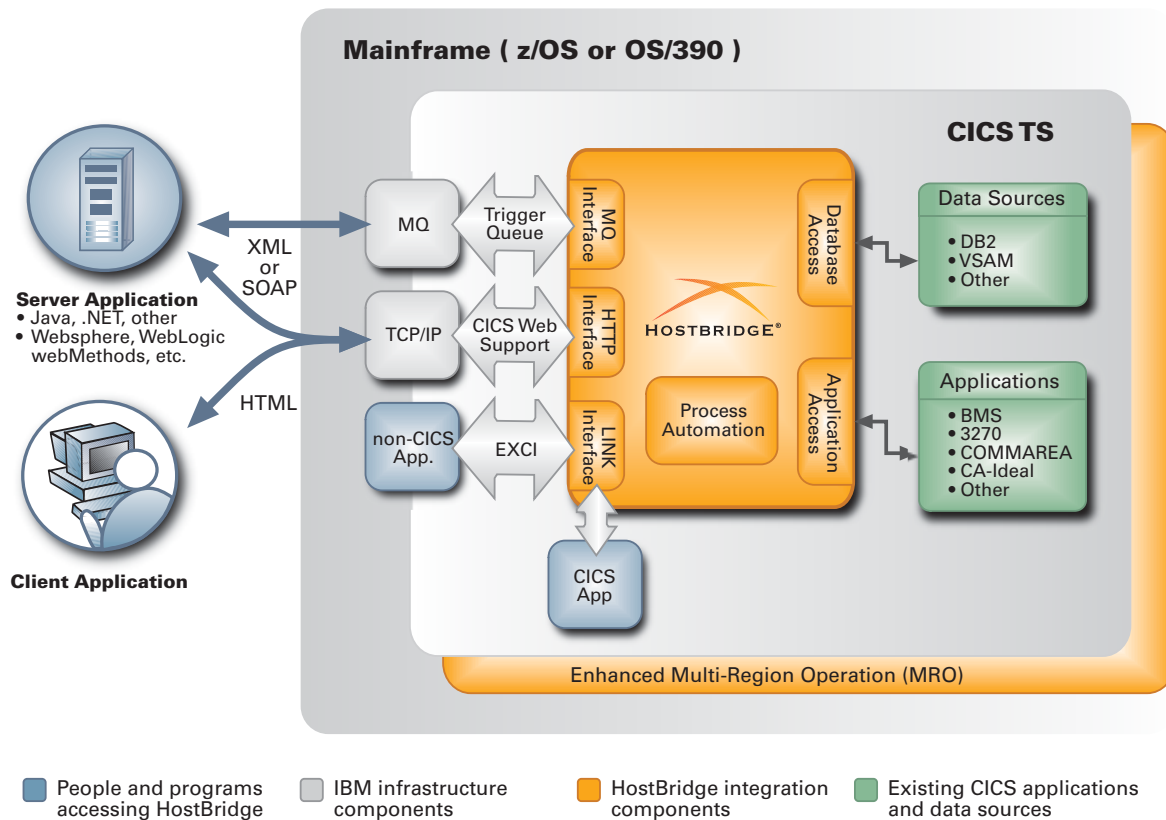


Figure 9. HostBridge overview

Figure 10 shows the basic model for accessing CICS applications as web services through HostBridge.

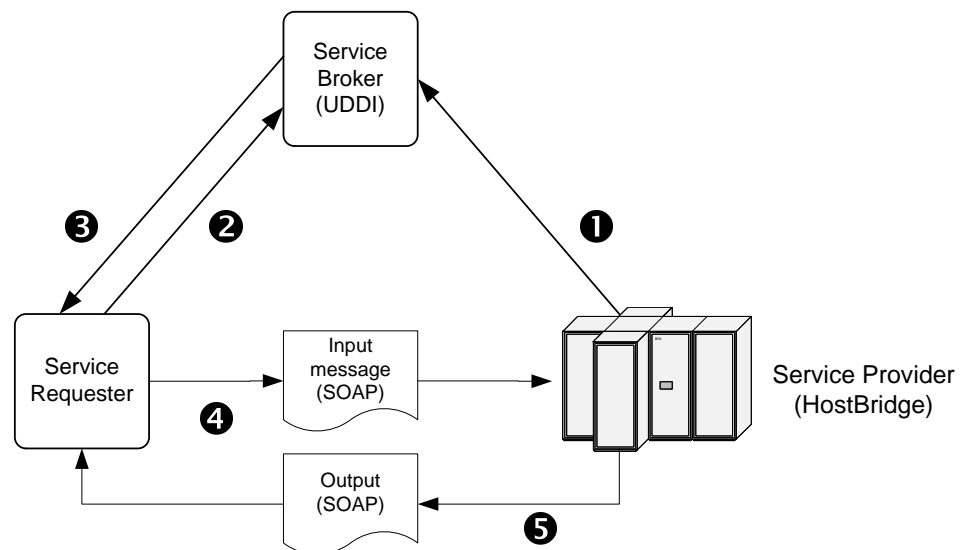


Figure 10. HostBridge and the web services model

The diagram in Figure 10 shows three web services: a Provider (HostBridge) that provides the web service, a Requester that uses a web service, and a Broker that finds Providers for Requesters. The following steps are required to find and use a web service. (Subsequent requests do not require steps 1-3.)

1. HostBridge uploads a WSDL specification to publish its web service with a Broker.
2. The Requester queries the Broker for a web service by name or category.
3. The Broker selects a Provider and returns the Provider information to the Requester.
4. The Requester uses the information from the Broker to format and send a SOAP message to the HostBridge.
HostBridge returns an XML document to the Requester with the CICS data enclosed.

The following examples illustrate how SOAP enables interoperability. In our examples, Figure 11 shows a basic XML document that HostBridge can receive as a request to begin a CICS transaction. The document tells HostBridge to use the Enter key to initiate the JMPO transaction without input data.

```
<?xml version="1.0" ?>
<hostbridge_input1>
  <transaction>
    <parameters>
      <trandid>jmpo</trandid>
      <aid>enter</aid>
    </parameters>
  </transaction>
</hostbridge_input1>
```

Figure 11. Sample XML request sent to HostBridge

To request the same transaction as a web service, an application would send a SOAP message to HostBridge as shown in Figure 12.

```
<?xml version="1.0" ?>
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <hb:hostbridge_input1 xmlns:"http://www.hostbridge.com/ns/">
      <hb:transaction>
        <hb:parameters>
          <hb:trandid>jmpo</hb:trandid>
          <hb:aid>enter</hb:aid>
        </hb:parameters>
      </hb:transaction>
    </hb:hostbridge_input1>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 12. Sample SOAP request sent to HostBridge

The SOAP message is an XML document, but the format is different. The basic XML document from Figure 11 is present, but it is “wrapped” in a pair of elements: <SOAP-ENV:Envelope> and <SOAP-ENV:Body>. According to the SOAP specification, SOAP messages wrap their contents in an envelope and use the body to identify the beginning and ending of the payload. The addition of the “hb:” prefix to the standard XML elements from Figure 11 illustrates the use of namespaces. Namespaces provide a tool for developers to use to identify which elements are HostBridge input elements.

Enabling CICS Applications as Web Services

Because HostBridge XML-enables the CICS transactions, they can be accessed as web services using the same methods used to integrate native web-based applications. The steps required allowing remote applications to invoke CICS transactions as web services using HostBridge are simple and straightforward.

1. XML-enable your CICS transactions with HostBridge.³
2. Set up the CICS transactions with HostBridge that you want to define as a web service.
3. Create a WSDL file to describe your web service.
4. Publish your WSDL file to an internal or external UDDI server so other web services can find your CICS web service.
5. Access the web service using SOAP, HTTP GET, SMTP, or any other standard protocol.

Conclusion

CICS Transaction Server includes facilities that allow third-party vendors to create connectors that can immediately enable legacy applications as web services. These facilities also provide additional benefits over gateways, such as improved performance and increased stability compared to their screen scraping counterparts. By using the CICS facilities, HostBridge makes it possible for applications to transparently invoke CICS transactions within web services architecture and receive the resulting data as well-formed XML.

For organizations that want to retain the value of their CICS applications, the combination of HostBridge, SOAs, and web services offers a practical and powerful integration solution. In the end, companies need to assess the value of the data contained in their CICS applications. Most companies have already determined that such data is highly valuable and they are looking for ways to preserve their investments. Given that recent surveys show the top strategic priorities of CIOs and CTOs is integrating systems and processes, the use of SOA and web services for legacy integration will grow rapidly.

Contact Information

For more information on HostBridge or to inquire about our free 30-day trial, please contact us using the information below.

Toll-free:
1.866.XML.CICS (965.2427)

International:
1.405.533.2900

Email:
info@hostbridge.com

³ See white paper entitled *XML-Enabling CICS Applications for e-Business* to see how you can use HostBridge in your organization.



100 E Seventh Ave, Stillwater, OK 74074
Ph. 405.533.2900
www.hostbridge.com