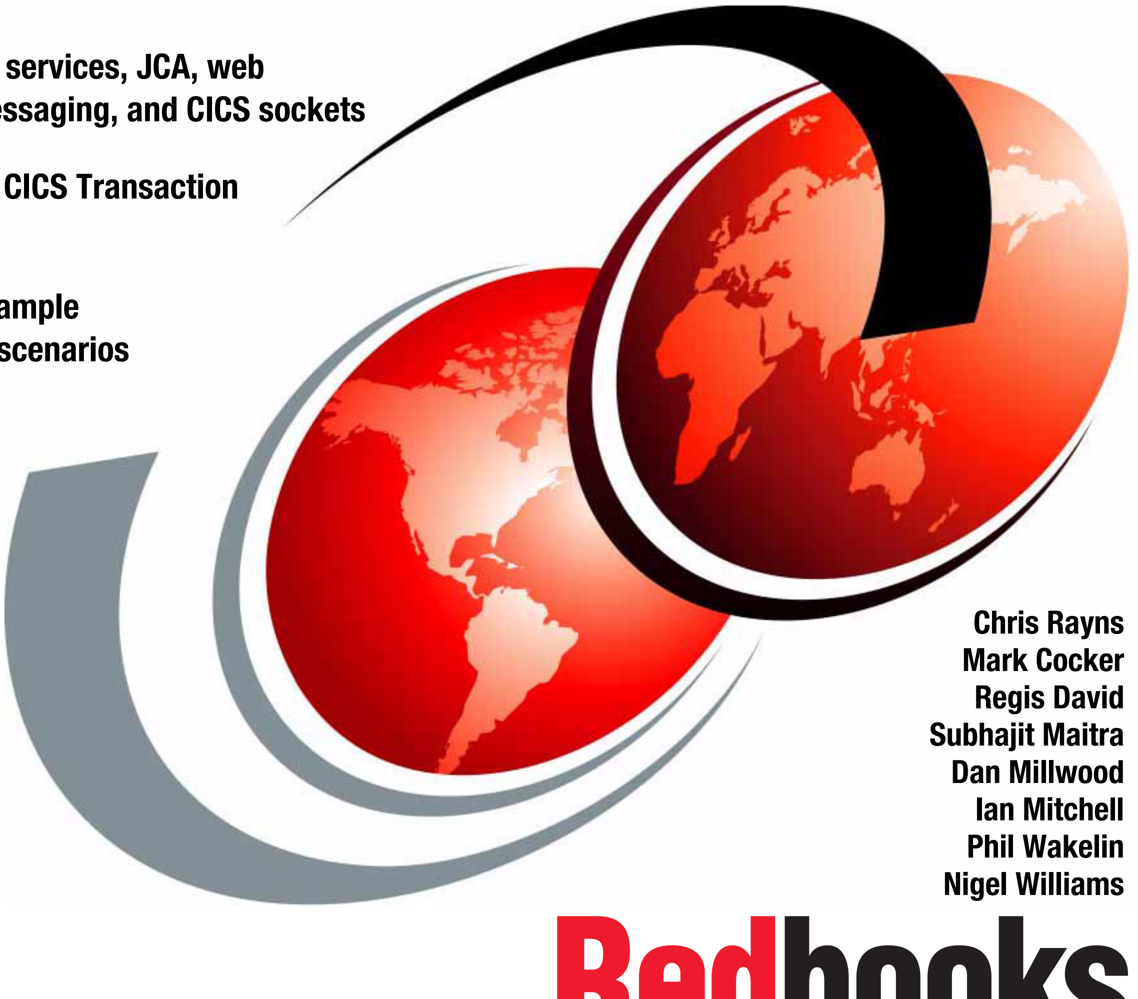
CICS and SOA
Architecture and Integration Choices

Covers web services, JCA, web support, messaging, and CICS sockets

Is based on CICS Transaction Server V4.2

Includes example integration scenarios



Chris Rayns
Mark Cocker
Regis David
Subhajit Maitra
Dan Millwood
Ian Mitchell
Phil Wakelin
Nigel Williams

Redbooks

ibm.com/redbooks

International Technical Support Organization

CICS and SOA: Architecture and Integration Choices

March 2012

SG24-5466-06

Note: Before using this information and the product it supports, read the information in "Notices" on page xi.

Seventh Edition (March 2012)

This edition applies to the CICS Transaction Server Version 4, Release 2.

Contents

Notices

Trademarks

Preface

The CICS® Transaction Server is used extensively for high-volume transaction processing. Because new IT solutions are rarely deployed on a single IT system and reuse of existing assets is a common goal, integration between CICS and other systems is a frequent requirement.

The IT industry has converged on a set of common standards and principles that have led to the emergence and maturity of the service-oriented architecture model (SOA) of IT integration. The SOA style of integration involves breaking an application down into common, repeatable “services” that can be used by other applications, both internal and external, in an organization, independent of the computing platforms on which the business and its partners rely.

In recent years CICS has added a variety of support for SOA and now provides near seamless connectivity with other IT environments. This IBM® Redbooks® publication is intended to help IT architects to select, plan, and design solutions that integrate CICS applications as service providers and requesters.

First we provide an introduction to CICS service enablement and introduce the architectural choices and technologies on which a CICS SOA solution can be based. We continue with an in-depth analysis of how to meet functional and non-functional requirements in the areas of application interface, security, transactional scope, high availability, and scalability. Finally, we document three integration scenarios to illustrate how these technologies have been used by customers to build robust CICS integration solutions.

The team who wrote this book

This book was produced by a team of specialists from around the world working at IBM UK Laboratories, Hursley, England.

The IBM Redbook team (back left to front right): Chris, Phil, Nigel, Ian, Dan, Subhajit, Mark, and Regis

Chris Rayns is an IT Specialist and Project Leader at the ITSO, Poughkeepsie Center in New York. Chris writes extensively on all areas of CICS TS and CICS TG. Before joining the ITSO, he worked in IBM Global Services in the United Kingdom as a CICS IT Specialist.

Mark Cocker is a Senior Software Engineer in the CICS Strategy and Planning team at IBM, based at Hursley Laboratory, England. Mark has 20 years of experience in CICS development, service, beta programs, and the IBM Design center. He holds a degree in information systems management from Bournemouth University and is an IBM Certified SOA Associate and Solution Designer - CICS Enablement for e-business. His areas of expertise include CICS TS and enterprise connectivty. He has written several CICS SupportPacs and papers, and he presents on CICS topics regularly at conferences.

Regis David is a Senior IT Product Services professional in France. He has 30 years of experience working on the full scope of the CICS ecosystem. His areas of expertise include advanced client/server implementations such as web services, Java Connector Architecture, and RESTful style, including messaging. He is an expert of pragmatic service-oriented architecture implementations. He runs multiple presentations in France, focussed on new technology adoption within CICS and System z®.

Subhajit Maitra is an Senior IT Specialist working for IBM Advanced Technical Support based in Hartford, CT. His areas of expertise are WebSphere Message Broker and WebSphere MQ on System Z. He has over 16 years of experience in information technology as a developer, designer, and architect on various projects. He has previously worked with the ITSO in building workshops and delivering them worldwide. He holds a master's degree in computer science from Jadavpur University, in Kolkata, India.

Dan Millwood is a Software Developer working within the CICS Transaction Server for z/OS® development team in England. For 17 years he has worked at IBM on messaging and transaction processing products. He holds a degree in computer science from Southampton University. His areas of expertise include WebSphere MQ, WebSphere Application Server, and CICS TS. In his role with CICS TS, Dan has focused on the integration of CICS TS with other products, using technologies such as web services.

Ian Mitchell is an IBM Distinguished Engineer with responsibility for the technical architecture of the CICS Product Portfolio. He has more than 20 years of experience as a technical leader in IBM Hursley and has been in the forefront of creating and introducing many important CICS innovations, including workload management, Sysplex support, business transaction services, Java and EJBs, web services, event processing, and the CICS Explorer®. In this role Ian has worked with many of the technical leaders across all the mainframe technology components. Ian also has close relationships with many of IBM's largest customers running critical systems using CICS and the mainframe.

Phil Wakelin works for IBM UK in Hursley and is a member of the CICS strategy and planning teaming. He has worked with many different CICS technologies for the last 20 years. He is currently responsible for new functionality in the areas of CICS interconnectivity and CICS Java support. He is the author of many white papers, SupportPacs, and IBM Redbooks publications.

Nigel Williams is a Certified IT Specialist working in the IBM Design Centre, Montpellier, France. He specializes in enterprise application integration, security, and SOA. He is the author of many papers and IBM Redbook publications, and he speaks frequently on CICS and WebSphere® topics.

Thanks to the following people for their contributions to this project:

John Burgess, Paul Cooper, Rob Jones, Steve Hobson, Colin Paice, and Pete Siddall
IBM UK Software Group, Hursley

Don Bagwell and Carolyn Elkins
IBM Advanced Technical Skills

Lennie Dymoke-Bradshaw and Michael Lowe
IBM UK Software Group

David Follis and Linwood Overby
IBM US Software Group

Thanks also to the authors of the previous edition of this book; Martin Keen, Chris Backhouse, Jim Hollingsworth, Stephen Hurst, Mark Pocock.

Now you can become a published author, too!

Comments welcome

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks
- Follow us on Twitter:

  http://twitter.com/ibmredbooks
- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm
- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Seventh Edition, March 2012

This revision has been significantly updated to include the additions and changes described below.

New information

- Features introduced in CICS TS 3.2 V4.1 and V4.2
- Features introduced in WebSphere MQ V7
- Features introduced in CICS TG V7 and V8
- Atom feeds
- WebSphere Optimized Local Adapters (WOLA)
- CICS sockets information added to chapters in Part 2
- High availability in Part 2
- Integration scenarios in Part 3

Changed information

- Part 2, “Qualities of service” on page 87, focus on the z/OS platform for CICS TG
- Part 2, “Qualities of service” on page 87, was restructured for consistency across the connectivity options
- Chapter 5, “Application interfaces” on page 89, now includes data conversion
- CICS EJB support information removed
- SOAP for CICS feature information removed

Sixth Edition, October 2006

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- CICS TS V3.1, CICS Web services support
- CICS Web services support customer scenario
- CICS Service Flow Feature

Changed information

- Chapter 1, “Introduction to Service Enablement for CICS” on page 3
- CICS TG V6 updates
- SOAP for CICS and CICS EJB support moved to an appendix

Fifth Edition, February 2005

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- SOAP for CICS feature customer scenario
- CICS TS V2.3, Link3270 bridge extended support
- Chapter 7, “Transactional scope” on page 185

Changed information

- SOAP for CICS feature added to all chapters
- J2EE Connector Architecture added to all chapters
- “Matters of State” chapter removed

Fourth Edition, October 2002

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- CICS TS V2.2, Link3270 bridge
- CICS TS V2.2 ECI over TCP/IP support
- CICS TS V2.3, Link3270 bridge extended support
- Miami-Dade County customer scenario

Changed information

- CICS TS V2.2 EJB tooling and security (Chapter 4, “Security and Chapter 7, Application development”)
- CICS TG V5 COMMAREA null stripping (Chapter 6, “Performance and scalability”)

Third Edition, July 2001

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- CICS TS V2.1 EJB support
- CICS to TCP/IP Sockets Interface
- Patterns for e-business

Changed information

- Application Development for CWS 3270 Web bridge
- Security considerations for CICS Web support (APAR PQ45098)
- Part 3 chapters updated and consolidated into the “CICS Web decision points” chapter in Part I

Second Edition, March 2001

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- “Matters of State” chapter

Changed information

- “Introduction to CICS and Web-enabling” chapter merged with “CICS/Web selection Guide” chapter
- “CICS Web Application development,” “Changes to Existing applications,” and “Portability” chapters consolidated into “Application Development” chapter
- “Administration” chapter removed
- NetCICS removed as a CICS Web solution
- Customer scenarios updated

Part 1

Architecture and technologies

In Part 1 we first position CICS as a platform for service enablement. We then discuss what architectural approaches and connectivity options are available to you to service enable your CICS applications. We focus on these solutions:

- Web services
- Java EE Connector Architecture (JCA)
- Web support
- Messaging
- TCP/IP sockets

For each strategic CICS access technology, we provide product information and an overview of the major components and deployment topologies. Finally, we review the specific options for reusing 3270-based CICS applications.

1 Introduction to Service Enablement for CICS

SOA is an integration architecture approach that is based on the concept of services. The business and infrastructure functions that are required to build distributed systems are provided as services that individually or collectively deliver application functions to either user applications or to other services.

By adopting an service-oriented architecture (SOA) approach and implementing it using supporting technologies, companies can build flexible systems that implement changing business processes quickly and can make extensive use of reusable components.

This chapter introduces the concept of SOA and discusses how it applies to CICS. This includes a discussion about the business value of SOA and its IT benefits, the advantages of transforming CICS assets into SOA solutions, and the evolution of services into Cloud computing.

1.1 SOA: An architectural approach

Business processes are changing faster and faster, and global competition requires the flexibility that SOA can provide. SOA can help achieve the best reuse of your existing IT investments, as well as the new services that you are developing today. SOA makes integration of your IT components easier by making use of well-defined interfaces between services. SOA also provides a flexible and secure architectural model for integrating services from business partners, customers, and suppliers into an enterprise business process.

To simplify the adoption of SOA, there are five common approaches, or entry points, which you could choose from to be the focus of your initial projects:

- People

  Collaboration-improving productivity by giving employees and partners the ability to create a personalized, consolidated way to interact with others.
- Process

  Model, optimize, and deploy processes on the fly and monitor the effectiveness of the altered processes.
- Information

  Improve business insight and reduce risk by using trusted information services delivered in line and in context.
- Connectivity

  Effectively integrate people, processes, and information in a secure environment with the flexibility to quickly adapt to changing business needs.
- Reuse

  Newly created and reusable services are the building blocks of SOA to deliver reduced cycle times and elimination of duplicate processes.

Rather than being a revolution, SOA is an evolution of best practices and technologies that have gone before. It takes advantage of developments in internet-based technology and interoperability standards to offer unrivalled IT benefits. There have been many definitions for SOA, but all lead to the concept of loosely coupled business services that are provided in an interoperable and technology agnostic manner. A useful definition of SOA is provided by the OASIS group:

> "A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations."
>
> `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm`

1.1.1 Basic components of an SOA

At the most basic level, an SOA consists of these components (Figure 1-1):

- Service provider
- Service requester
- Service registry

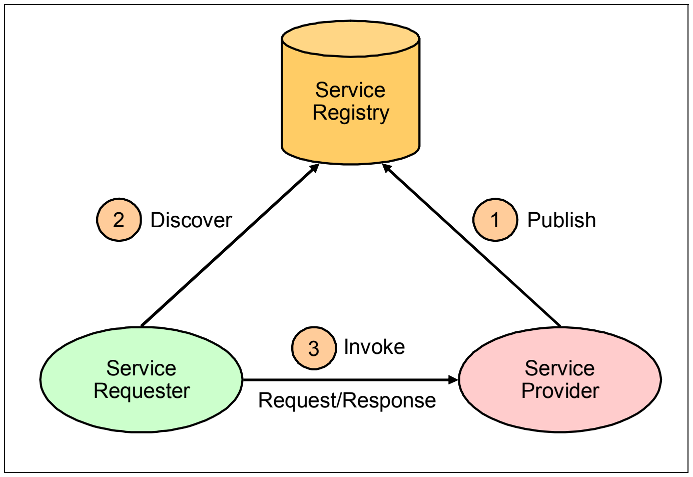


Figure 1-1 SOA components and operations

Here are more details about each component:

- The *service provider* implements a service, and in some cases publishes its interface and access information to a *service registry*. Each provider must decide which services to expose, evaluate trade-off between security and easy availability, determine how to price the services, or figure out how to exploit the value of the services if they are free. The provider also has to decide in what category the service should be listed, and what sort of trading partner agreements are required to use the service.
- The *service registry* is responsible for making the service interface and implementation access information available to service requesters. There are public service registries available over the internet to an unrestricted audience, as well as private service registries that are only accessible to users within a company-wide intranet.
- The service requestor *discovers* entries in the service registry and then binds to the service provider to invoke the defined service.

Each component can also act as one of the two other components. For instance, if a service provider needs additional information that it can only acquire from another service, it acts as a service requester.

1.1.2 Defining a service

SOA is an architectural approach to defining integration architectures that are based on the concept of services. A service can be described as a function that can be offered or provided to a requester. This function can be an atomic business function or part of a collection of business functions that are wired together to form a process.

There are many additional aspects to a service that must also be considered in the definition of a service within an SOA. These are the most commonly agreed-on aspects of a service:

- Services encapsulate a reusable business function.
- Services are defined by explicit, implementation-independent interfaces.
- Services are invoked through communication protocols that stress location transparency and interoperability.

Reusable business functions

Ideally, a service should be reusable and thus be accessible by more than one requesting application. For example, a service that offers a calculation such as an insurance quote could be used by requesters inside the enterprise and by third parties, as long as the interfaces of the component that offers the service are defined clearly.

Clearly defined interfaces

The service interface should encapsulate only those aspects of process and behavior that are used in the interaction between the requester and the provider. An explicit interface definition, or contract, binds a service requester with the provider. The interface should specify only the mutual behavior that is required for the interaction and nothing about the actual implementation of the requester or provider. This arrangement means that those system aspects where the requester and provider are hosted (their platforms) are independent of the interaction and are free to change. This abstraction allows for flexible improvements to the underlying IT infrastructure.

Location transparency

SOA does not specify that the requester needs to use any specific protocol to have access the service provider. A key principle in SOA is that a service is not defined by the communication protocol that it uses, but instead, should be defined in a protocol-independent way that allows different protocols to be used to access the same service. Ideally, a service should only be defined once, through a service interface, and should have many implementations with different access protocols. This type of definition helps to increase the reusability of any service definition.

1.2 Web services

Web services are now the de-facto standard for implementing a basic SOA. Web services take advantage of existing open-standard web technologies, such as XML, Uniform Resource Locator (URL), and Hypertext Transfer Protocol (HTTP), that are themselves a set of standards that facilitate open system-to-system communication.

High ceremony versus low ceremony

Many of the most mature and rich SOA implementations use the full range of agreed-on formal standards that have been created by bodies such as W3C and OASIS. They use Web Services Description Language (WSDL) to provide a rich description of the services, a service registry to provide publication and governance for their services, and SOAP as their service invocation mechanism. Added to this basic framework, they might make use of additional capabilities specified in standards such as WS-Atomic Transaction or WS-Security. In demanding such a rich style of implementation, an organization will be demanding a *high ceremony* approach to the creation and maintenance of such services.

Other situations will not demand such high ceremony processes to manage the services. The emergence of *RESTful* services over the internet is in recognition of those situations and provides a *low ceremony* alternative for service enablement. RESTful systems rely more on the basic convention of the web than on additional technologies and standards to provide services or define interfaces.

An example of a difference between high ceremony, WSDL/SOAP-based implementations and a low ceremony RESTful approach is the relative lack of a formal artifact that describes the “shape” of a service. Systems employing RESTful principles still require a means to locate services, but rather than use a specialized registry, rely on existing common means used to locate resources identified by URL. Similarly, a RESTful system forgoes the richness of a bespoke vocabulary of operation names demanded in WSDL and leans on the four HTTP verbs:

- GET
- PUT
- POST
- DELETE

A service based on an Atom API is such a RESTful system and is often used to provide a simple HTTP-based protocol for creating and updating web resources. For more detail see 2.4.3, “HTTP and Atom feeds” on page 31.

1.3 CICS as a platform for service enablement

CICS Transaction Server has many features that make it an excellent platform to host applications that can be enabled as services. One of the key decisions when enabling CICS applications as services is whether to use a connector model (indirect connection) or to host web services directly inside CICS (direct connection). Figure 1-2 illustrates these two models, where A represents the *adapter*, I the *integration logic*, B the *business logic*, and D the *data access logic*.

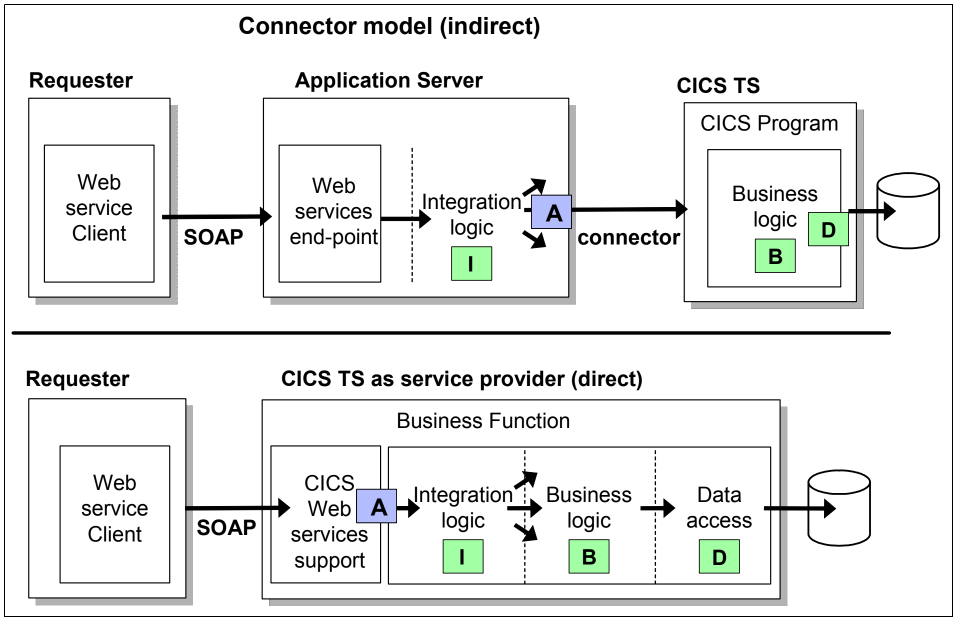


Figure 1-2 Direct or indirect models for CICS Web service enablement

The choice of architectural approach is a key decision because it might affect the costs of developing applications and their long-term value. Business factors, such as the availability of skills, might be just as significant as technical factors influencing this decision. It is important to recognize that there is no single right answer, just as there is no right programming language for all applications. Some technical factors to take into account include application interfaces, security, transactional scope, availability, and scalability characteristics. These are discussed in more detail in Part 2, “Qualities of service” on page 87.

1.3.1 Connector model

In the connector model, the service endpoint is hosted outside of CICS, for example, in WebSphere Application Server. The adapter is deployed in the application server. It transforms the incoming request message into a COMMAREA or container that is then passed to the CICS application using a connector. The CICS Transaction Gateway and the WebSphere Optimized Local Adapter are two examples of connectors that can be used in this model.

When using the J2C connector technology from Rational® Application Developer, all the adapter code is generated to access fields within the CICS COMMAREA and perform necessary conversion and data formatting before invoking the CICS program via the connector. These tools can also generate code to easily expose the adapter as a web service, thereby enabling a CICS program to be exposed as a web service in an application server via the CICS TG with little if any coding.

1.3.2 Direct model

In the direct model, the service endpoint is hosted in CICS. The adapter is deployed within CICS, and it transforms the incoming request message into a COMMAREA or container that is then passed to the CICS application.

CICS Web services support enables a CICS program to be a SOAP web service provider or service requester. The definition of the SOAP service is defined in a WSDL file. Typically, tools are used to import the WSDL file and generate a proxy for the web service client to use to construct and send the SOAP message.

Implementing SOAP-based services naturally encourages a more mature definition of those services than previously available integration and connectivity mechanisms. This maturity is supported by the use of governance and life-cycle management tools such as WebSphere Service Registry and Repository (WSRR) and Rational Asset Manager (RAM).

SOAP is an XML-based protocol and as such provides ample features that simplify the handling of the messages. SOAP messages are easy to work with and, combined with schemas, the message content becomes self-describing. However, compared to a pure-binary message, a SOAP message might appear verbose and inefficient. This is the price for loose-coupling and simplicity.

Different interaction patterns demand either synchronous or asynchronous transport of requests and replies, and CICS supports web services using SOAP over both HTTP for sync and WebSphere MQ primarily for async.

It is also true that different situations give rise to more or less complexity in the information flowing into and out of a service. In simple cases, the reduced complexity can be leveraged with a deployment solely involving CICS capabilities and a natural, explicit flow of information. As complexity and/or volume in each request increases, other techniques can be used to deal with the complexity of the information. For example, Message Transmission Optimization Mechanism (MTOM) and XML Optimization Protocol (XOP) can be used to reduce the volume of a complex or verbose message by changing its encoding from relatively readable XML to opaque base64.

At the extreme, all of the loose-coupling advantages might be outweighed by the expense of dealing with large, verbose SOAP messages. Reverting to a (probably private and proprietary) binary format is more resource-efficient, but requires both the requester and the provider to agree on the non-standard form and will likely be much less flexible to change.

Over and above the highly useful transport-level security provided by SSL, message or element-based security can be used in the situation that demand a finer-grained control of message content authentication and privacy.

While the principles of loose-coupling and service encapsulation would discourage the need to create distributed transactions across service provider boundaries, as a transaction monitor, CICS supports the web services two-phase commit protocol specified by web services – Atomic Transactions (WS-AT).

Additional service enablement features in CICS TS V4

While many business-critical, system-to-system interaction scenarios demand the level of contractual agreement enabled by a WSDL-described SOAP-based message exchange, other situations can be best served with a lighter weight, more flexible approach. When implementing a service provider, the principles of Representational State Transfer (REST) can be applied in these cases, and many CICS applications can fit this model. A RESTful interface is often a relatively literal reflection of the underlying data, with the HTTP verbs (GET, PUT, POST, DELETE) being used to implement the common operations create, read, update, delete (CRUD) on business objects identified simply by a Uniform Resource Locator (URL).

A good example of a RESTful interface is the Atom protocol, and CICS TS V4 provides support to help you quickly and easily expose CICS-managed data to a wide variety of requesters using Atom. Atom is an excellent option when requesters need to find and filter sets (feeds) of information (articles), and then work with the information.

Service enablement in its wider context can extend to interaction patterns beyond the familiar request-reply exchange of information. As systems become more complex and demands for agility increase, an event-based paradigm can help to expose information from your existing systems more quickly. Many events of interest to an enterprise are likely to be happening inside CICS applications, but the application's design (not to mention its implementation) perhaps did not anticipate the value of those events, and so did not include a means to expose them.

The event processing capabilities in CICS TS V4 enable non-invasive publication of such useful events from existing CICS applications, another example of reusing the applications that you already have in an adaptable and agile way. Basic business events from the application and the system can be efficiently captured, filtered, augmented, formatted, and published without writing additional code. WebSphere MQ is an ideal transport to route the events and associated data to consumers such as WebSphere Business Events or WebSphere Business Monitor, which are optimized to perform sophisticated event correlations from multiple sources and provide business dashboards.

1.4 CICS TS as a platform in the cloud

Cloud computing is a model for enabling convenient, on-demand network-based access to a shared pool of configureable computing resources such as service-enabled applications. These resources can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model builds on top of the service enablement of applications and promotes additional availability and flexibility. Cloud applications are composed of five essential characteristics:

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured services

Many of these characteristics are provided by System z and can be readily exploited when deploying applications as CICS services, for instance, resource measurement via CICS monitoring, broad network access via TCP/IP or SNA connectivity, rapid elasticty through CICS open transaction environment, and System z capacity on demand.

There are three typical deployment models for cloud services (Figure 1-3):

- Private cloud

  The cloud infrastructure is operated solely for an organization.

- Public cloud

  The cloud infrastructure is made available to the general public or a large industry group.

- Hybrid cloud

  The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities.

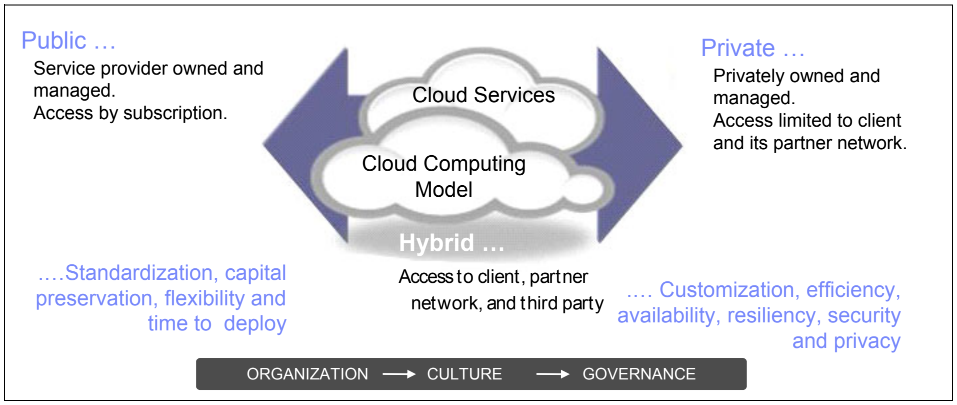

Public …
Service provider owned and managed.
Access by subscription.
Cloud Services
Cloud Computing Model
Private …
Privately owned and managed.
Access limited to client and its partner network.
Hybrid …
Access to client, partner network, and third party
….Standardization, capital preservation, flexibility and time to deploy
…. Customization, efficiency, availability, resiliency, security and privacy
ORGANIZATION → CULTURE → GOVERNANCE

Figure 1-3 Cloud computing model

The private cloud is the model that is most widely used today for CICS-based services, as enterprise computing resource are typically privately owned by corporations. However, CICS outsourcing organizations do provide shared services that are a form of community cloud installations.

In addition, services provided to the consumer can be viewed as being in one of three models as regards the capability provided to the service consumer:

- Software as a Service (SaaS)

  Usage of the provider's services running on a cloud infrastructure

- Platform as a Service (PaaS)

  Deployment of services onto a cloud infrastructure

- Infrastructure as a Service (IaaS)

  Provisioning of resources within a cloud infrastructure, where the consumer is able to deploy and run software

For further details on Cloud computing refer to the following website:

http://www.nist.gov/itl/csd/cloud-020111.cfm

1.5 The modern CICS management experience

Configuring and maintaining modern CICS business service implementations demands comprehensive management facilities. With the majority of enterprises running many tens or hundreds of CICS regions to support their needs for capacity, resilience, and workload management, the capabilities of CICSPlex® Systems Manager (CICSPlex SM) are ever more valuable. With the ability to provide unified application-oriented or system-oriented views across your regions and effectively take action, CICSPlex SM is relied upon as part of the management scheme for many customers, both large and small.

Today's users are demanding a modern experience that closely matches what they encounter in daily use of other IT or consumer systems, not the old-fashioned green screen experience so often associated with mainframes. The CICS Explorer provides that experience starting with read-only resource access in CICS TS V3 and full-function administration support in CICS TS V4.

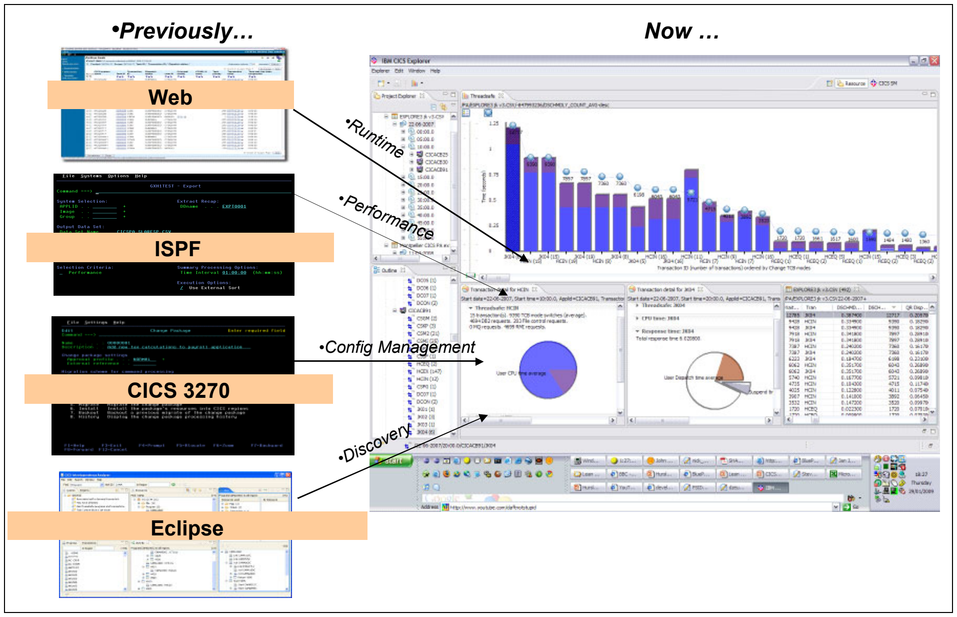


Figure 1-4 CICS Explorer: The new face of CICS

An example of how new functions in the CICS management experience help systems run more smoothly is CICS Transaction Tracking, which is provided in CICS TS V4.2. Supplementing end-to-end tracking capabilities provided by Tivoli®, transaction tracking in CICS enables fine-grained questions to be answered about related execution resources. Two common scenarios are to trace back the thread of tasks related to a service request to its point of origin across several CICS regions and to trace forward the tasks that are serving requests from a particular listener or terminal owning region, both of which are supported with dedicated transaction tracking views in the CICS Explorer.

In the days of simple green screen 3270 applications, it was relatively simple to comprehend and manage the resources required to support a given application. With today's wider variety of access methods and resources, the range of definitions needed could make deployment more complex. CICS employs both its own resource bundles and those from OSGi to help reduce this complexity.

CICS bundles support the definition, deployment, and management of a set of related resources, including OSGi bundles for Java programs, event bindings, and Atom feeds enabling the life cycle of disparate resources to be managed through operations on a single CICS BUNDLE resource.

For the growing number of Java components being used in CICS TS, OSGi (implemented in the JVM server environment in Version 4.2) is the industry standard way to package components for server deployment. OSGi bundles include meta-data, which enables the server to manage installing, enabling, disabling, and discarding the application classes without requiring disruptive outages of the server. Part of the management is to ensure that the inter-component dependencies expressed in the meta-data of the bundles are respected, eliminating the chance of mismatches while serving requests.

2

Architectural choices

This chapter proposes an architecture for your CICS applications and a set of requirements and connectivity options that influence how to best service enable CICS assets into an SOA. We focus on the modern connectivity options:

- Web services
- JCA
- HTTP and ATOM feeds
- Messaging
- TCP/IP sockets

When deciding on the best integration solution, consider these questions:

- What interface and granularity is most appropriate for the service requester?
- What interface does the CICS application already provide?
- Is a synchronous or asynchronous interaction pattern more appropriate?
- What are the end-to-end security requirements?
- Should the CICS application have its recoverable resources synchronized with the service requester?
- What are the availability and scalability targets?

2.1 CICS application architecture

It is a good general practice to clearly distinguish application components containing adapter, integration, business, and data access logic and implement these in separate layers with clearly defined interfaces:

- The *Client*, or service consumer, initiates the request. Examples include a web service requester, web browser, WebSphere MQ client, TCP/IP socket client, 3270 device, z/OS batch program, and other CICS applications.
- The *Adapter layer* processes the protocols and data with the client, establishes the transaction and security context in CICS, and interfaces to the integration or business layer.
- Optionally when required, the *Integration layer* implements a sequence of calls to business logic for situations where it is more efficient, better encapsulation, or simply makes the services easier to consume if done in CICS rather than the client making several calls directly to the business layer.
- The *Business layer* implements the service.
- The *Data access layer* updates DB2®, VSAM, IMS™, or other resources.

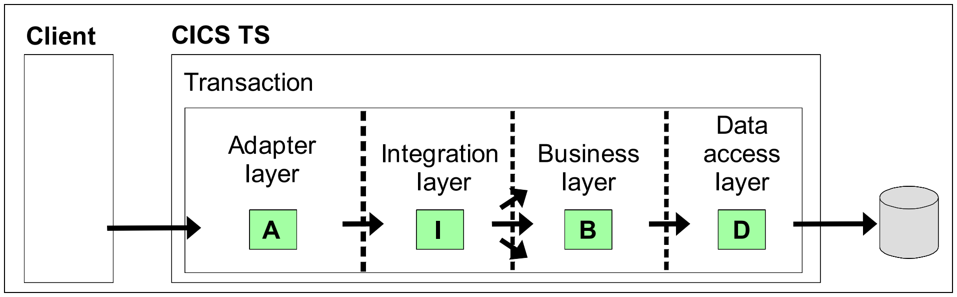


Figure 2-1 CICS application architecture layers

Architecting the CICS application into these layers provides the best opportunity for reuse by many types of clients today and into the future, and provides an important opportunity for workload management, high availability, and isolation.

The interface between the layers ideally would be made using the CICS LINK command. The data is passed with either of these:

- A channel

  A named holder of one or more *containers*. Both the channel and containers have names, have an easy-to-use CICS set of commands, and are automatically scoped to the task. A container can hold either text data in one of the many supported encodings such as EBCDIC, UTF, or ASCII, or binary data such as a COBOL copybook, ‘C’ structure, or a photograph. The size of each container is only limited by the storage in the CICS region, so it can potentially be up to hundreds of megabytes. Together these features provide a very capable and flexible mechanism to implement the interface.

- A COMMAREA

  A single binary data area limited in size to 32 KB. This was used by CICS applications for many years before the channel interface was available.

CICS also supports the COBOL CALL statement for one program to interface to another, which can be more efficient at run time than the CICS LINK command. However, the target program is required to run in the same CICS region and is therefore only suitable when interfacing between programs within a layer.

2.2 Access method architecture

At a high level the access method used by a client consists of a combination of these:

- Adapters
- External connectors

An *adapter* is a program that accepts a request and converts the data from an external interface to the internal interface used by the CICS application. Figure 2-2 shows how a 3270 client, a browser, and a web service requester can access the same integration layer in the CICS application by using different adapters.

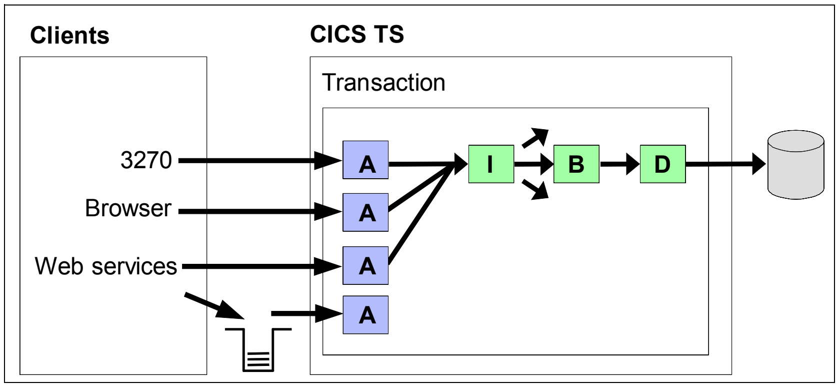


Figure 2-2 Connecting to adapters to reuse a CICS application

The 3270 client interacts with an adapter that uses the CICS RECEIVE command to obtain data from the screen and about the client before re-formatting it and calling the integration layer interface. The browser client ineracts with an adapter that uses the **`CICS WEB`**, **`EXTRACT WEB`**, **`EXTRACT TCPIP`**, and **`EXTRACT CERTIFICATE`** commands to receive the HTML form and data about the client before reformatting it and calling the integration layer interface. The web services requester interacts with an adapter that receives the data in a channel and containers before reformatting it and calling the integration layer interface. For more details see 5.4, “Adapters” on page 101.

An *external connector* provides the client a remote call interface and implements a private protocol to invoke the CICS application. The client uses interfaces with the external connector to create the format required by the CICS application in a channel and containers or COMMAREA. An example of an external connector is CICS Transaction Gateway, which implements the Common Connector Interface (CCI) specified by the Java EE Connector Architecture (JCA) and is used with adapters implemented as Java beans.

2.3 Application integration requirements

The requirements that affect how best to service enable a CICS application and integrate it into an SOA solution are introduced in the following areas:

- Application interface
- Client to server coupling
- Synchronous or asynchronous invocation
- Security
- Transactional scope
- High availability and scalability

2.3.1 Application interface

An application interface typically includes a service operation, an input and output message, a message format, a message exchange pattern, and a transport protocol. As a service provider, you will need to consider what set of application interfaces are required to most efficiently fulfill the needs of the service requesters while also providing flexibility for future reuse.

If you plan to reuse a CICS application, it might be that those interfaces can be reused without change. If not, you can develop new adapters or programs in the integration layer to interface with the existing CICS application in the business layer. For example, a service requester requires details of all outstanding orders for a customer. The the existing catalog inquiry programs only return a single order, so you decide to develop a new program for the integration layer to call the existing inquiry program in a loop to aggregate all the orders into a single response to the service requester. For more details see Chapter 5, “Application interfaces” on page 89.

2.3.2 Client-to-server coupling

Some access methods are described as *tightly coupled*, while others are described as *loosely coupled*. Tight coupling implies that the client and server share many assumptions and dependencies. Loose coupling is not a precise concept, but refers to several possibilities:

- Self-describing interfaces. That is, the service operations, messages, formats, exchange pattern, and transport protocols are fully described.
- Implementation independence. That is, the client might use dissimilar technology from the server program, thus offering operating system, middleware, and programming language independence.
- Server location independence. That is, the client program does not need to be changed when the location of the server program changes.

2.3.3 Synchronous or asynchronous invocation

The majority of access methods support synchronous invocation, meaning that a client request receives a single reply from CICS and the client waits for the reply. In the alternative approach, known as asynchronous invocation, the client request is not responded to immediately and the client might continue processing before receiving the response. A polling or event-based mechanism is required so that the response can be obtained at a later time. In some cases, an immediate response is sent from the server to the client, confirming receipt of the original request, therefore indicating that the application response will be delayed.

In general, asynchronous access solutions are more robust. Planned or unplanned outages, software upgrades, and other operational events have less impact on a client’s ability to send requests. Synchronous access solutions, however, normally offer better interoperability, for example, closer coordination of transactional updates.

2.3.4 Security

The first security requirement to consider is how end users and middle tier servers will be authenticated. Simple user ID and password authentication is still widely used, although SSL client certificates, Kerberos tickets, and other schemes are becoming popular. Whichever technique is adopted, the user's credentials must eventually be mapped to an external security manager (ESM) user ID to support the authorization and accounting requirements that normally apply to CICS applications.

The security characteristics of different CICS SOA access solutions are described in detail in Chapter 6, “Security” on page 133.

2.3.5 Transactional scope

This requirement refers to the capability of a given access option to support these types of transactions:

- *Local* transactions (one-phase commit), enabling a number of updates performed by CICS applications to be processed as a single unit of work
- *Global* transactions (two-phase commit), enabling an external server to coordinate updates performed by CICS with updates to local resources held by that server

The web services and JCA architectures both support global transactions. CICS provides support for the WS-Atomic Transaction specification, allowing us to tie together a client transaction and the invoked CICS transaction.

The transactional characteristics of different CICS SOA access solutions are described in detail in Chapter 7, “Transactional scope” on page 185.

2.3.6 High availability and scalability

Response time and CPU cost per transaction are important aspects of performance in a production system. CICS seeks to minimize these and is highly optimized for traditional styles of access, such as 3270 terminal access over a System Network Architecture (SNA) network. Most SOA solutions require additional elements such as connectors, adapters, and encrypted data flows. These are less optimized, and impose an overhead on the execution of the target business program.

Workload management is the process of distributing multiple requests for work over the resources that can do the work. It optimizes the distribution of processing tasks, therefore improving performance, scalability, and reliability of an application. It also provides failover when servers or systems are not available. The performance and workload management characteristics of different CICS SOA access solutions are described in detail in Chapter 8, “High availability and scalability” on page 219.

2.4 Integration options

This section introduces the major architectures that can be used to build CICS SOA solutions. Solutions based on these architectures benefit from the comprehensive set of development tooling that is provided to help in the generation of applications.

Standard architectures such as web services and JCA provide built-in support for qualities of service such as management of security and transactions. These qualities are slowly being introduced into the web services architecture. For example, the WS-Security specification, which provides for message-level security, and the WS-Atomic transaction, which provides two-phase commit transactional functionality, are now standard.

Standard transports are suitable for use by applications that require greater control of the protocol and that do not need the development tools that are generally provided by the standard architectures. These applications will also normally assume more responsibility for security, transactions, and recovery.

2.4.1 Web services

Web services is an implementation of a service-oriented architecture. A *service* is an application component that has a well-defined published interface that allows other application components to invoke operations on the service without any knowledge of how the service is implemented.

The technologies that can be used to implement a web services solution have received wide acceptance as the strategic way of building distributed IT solutions that integrate heterogeneous applications over the internet and intranets.

The web service specifications are completely independent of programming language, operating system, and hardware in order to promote loose coupling between the service requester (or consumer) and service provider. The technology is based on open standards such as these:

- eXtensible Markup Language (XML).
- SOAP: A standard protocol for exchanging XML messages.
- Web Services Description Language (WSDL), which defines an XML grammar for describing web services.\
- Universal Description, Discovery and Integration (UDDI), a registry mechanism that can be used to look up web service descriptions

Using open standards provides broad interoperability among different vendor solutions. These principles mean that companies can implement web services without having any knowledge of the service requesters, and vice versa. This facilitates just-in-time integration and allows businesses to establish new partnerships easily and dynamically.

Figure 2-3 shows how a SOAP message consists of an envelope containing zero or more headers and a body. Application designers determine the contents of the headers. The SOAP specification itself does not define what headers should be used. For example, application designers might define a header that contains authentication credentials or information for transaction management. The body is where the main end-to-end information (the payload) conveyed in a SOAP message must be carried. This information might be parameters for calling a service (for a service request) or the result of calling the service (for a service response).

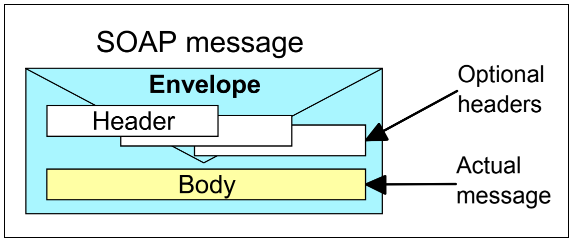


Figure 2-3 SOAP message

These are the major advantages of SOAP:

- It provides a standard for exchanging data in XML format, for example, the parameters used in a program call (for the inbound message) and the data resulting from the call (for the outbound message).
- It is protocol, platform, operating system, and programming language independent.
- It is flexible and extensible.
- It enables the use of web services standards such as WS-Security.

SOAP supports the remote procedure call (RPC) style of web service, in addition to the document message style. Although it is transport protocol independent, HTTP is the most widely used protocol today for transporting SOAP messages.

A web service is fully defined in a WSDL file. Most major environments that host applications have development tools that use the WSDL file to generate easy-to-use proxies or adapters to send and receive SOAP messages on behalf of applications.

Application programs running in CICS TS can participate in a heterogeneous web services environment as service requesters, service providers, or both, using either an HTTP transport or a WebSphere MQ transport. Figure 2-4 shows an outline of the web services support in CICS.

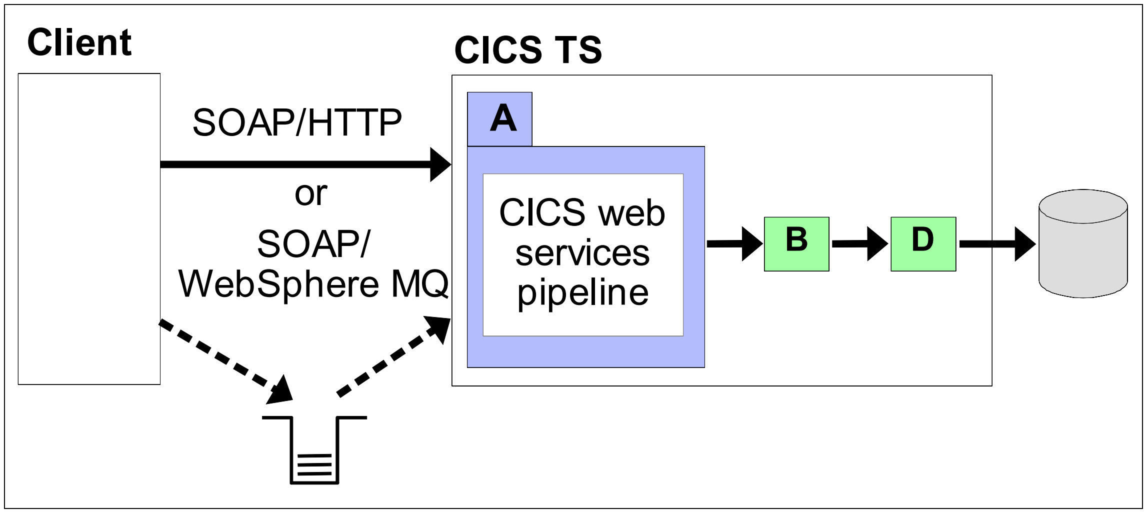
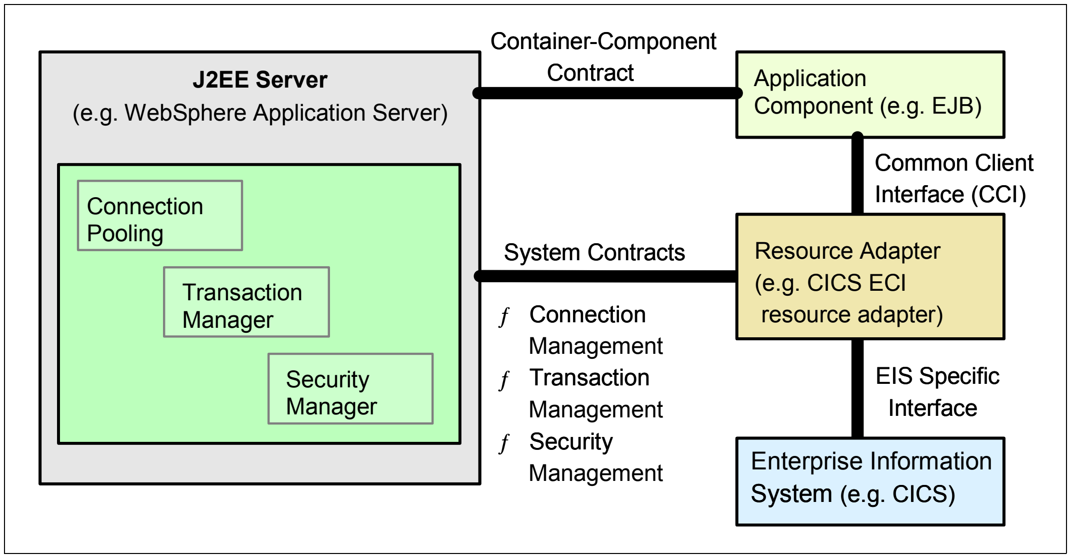


Figure 2-4   Connecting using CICS Web services

Refer to 3.1, "CICS Web services" on page 38, for more information about using CICS Web services.

2.4.2 JCA via CICS Transaction Gateway and WOLA

The Java Enterprise Edition (JEE) Connector Architecture (JCA) defines a standard set of APIs and interfaces for connecting from the JEE platform to heterogeneous Enterprise Information Systems (EIS). The JCA standards enable vendors such as IBM to provide a JCA *resource adapter* to connect and call services in an EIS such as CICS. Figure 2-5 shows the components in the JCA.



J2EE Server
(e.g. WebSphere Application Server)
Connection Pooling
Transaction Manager
Security Manager
Container-Component Contract
Application Component (e.g. EJB)
Common Client Interface (CCI)
System Contracts
ƒ Connection Management
ƒ Transaction Management
ƒ Security Management
Resource Adapter (e.g. CICS ECI resource adapter)
EIS Specific Interface
Enterprise Information System (e.g. CICS)

Figure 2-5 JEE Connector Architecture components

Figure 2-5 shows the JCA being used in a managed environment. That is, the application is running in a JEE environment such as WebSphere Application Server. In this case, management of connections, transactions, and security is managed by the application server. The JCA can also be used in a non-managed environment, in which case the application must manage connections, transactions, and security itself.

Note: We strongly recommend that you use a managed environment over a non-managed environment. The application development costs for a non-managed environment are significant and the quality of service is generally not as good as that provided by a managed environment such as WebSphere Application Server.

The Common Client Interface (CCI) defines a common application programming model for interacting with resource adapters and is independent of any specific EIS. Of course, this does not mean that a developer can write exactly the same code to access one EIS (for example, CICS) that he writes to access another EIS (for example, an IMS system). However, the generic CCI classes are the same in that they are independent of the EIS, whereas specific EIS classes cater to the differences. For example, the parameters used to call a CICS program are different from those used to invoke an IMS transaction, but the programming model is the same (independent of the EIS). As a result, you can increase developer productivity when developing applications to communicate with multiple EISs. The CCI programming interface is similar to other JEE interfaces, such as the Java Database Connectivity (JDBC) interface or Java Message Service (JMS) interface.

Resource adapters

The CICS Transaction Gateway (CICS TG) provides two resource adapters supporting the External Call Interface (ECI) and the External Presentation interface (EPI):

cicseci.rar	The CICS ECI resource adapter is provided with both CICS TG for multiplatforms and CICS TG for z/OS. It supports the XA and LocalTransaction interfaces.
cicsepi.rar	The CICS EPI resource adapter is provided only with CICS TG for Multiplatforms and provided access to 3270-based CICS transactions.

The ECI resource adapter is the simplest to use and the most commonly used CICS TG resource adapter. Support is provided both for synchronous and asynchronous calls. However, asynchronous calls using the CICS ECI resource adapter have their limitations. For example, you cannot make several concurrent calls and then wait for the response. You must take the response of each previous call before making another call.

The JCA resource adapters provided by the CICS TG are effective replacements for the CICS TG base Java classes. Support for the ECI resource adapters is included in the Rational Software Development Platform series of products, whereas tooling support for direct use of the ECI Java classes is not.

WebSphere Application Server on z/OS provides a WebSphere Optimized Local Adapter (WOLA) resource adapter that can call services in CICS providing that they are in the same z/OS LPAR.

System contracts

The JCA defines a standard set of system-level contracts between a JEE application server and a resource adapter. The standard contracts include these:

- A *connection-management* contract that provides a consistent application programming model for connection acquisition and enables a JEE application server to pool connections to a back-end EIS. This leads to a scalable and efficient environment that can support a large number of components requiring access to an EIS system.
- A *transaction-management* contract that defines the scope of transactional integration between the JEE application server and an EIS that supports transactional access. This contract allows a JEE application server to function as a transaction manager, and control two-phase commit transactions across multiple resource managers (known as global transactions). This contract also supports the LocalTransaction interface, which refers to one-phase commit transactions that are managed internally to a resource manager without the involvement of an external transaction manager. When using the managed environment with JCA 1.6, the transaction level (LocalTransaction or XA) can be specified on a connection basis using a connection factory property.
- A *security-management* contract that enables secure access to an EIS. This contract provides support for a secure application environment, which reduces security threats to the EIS and protects valuable information resources managed by the EIS. Both container-managed sign-on (in which the JEE application server is responsible for flowing security context to the EIS) and component-managed sign-on (in which the application is responsible for flowing security context to the EIS) are supported.

  When used with WebSphere Application Server for z/OS, the CICS ECI resource adapter enables automatic propagation of security credentials from the application server to CICS. This functionality is known as *thread identity support*.

These system contracts are transparent to the application developers, which means that they do not have to implement these services themselves. In a managed environment it is these system contracts that make the JCA such a powerful solution for integrating existing CICS applications with new JEE applications running in an application server such as WebSphere Application Server. Figure 2-6 shows how the CICS TG enables SOA access to a CICS business logic program.

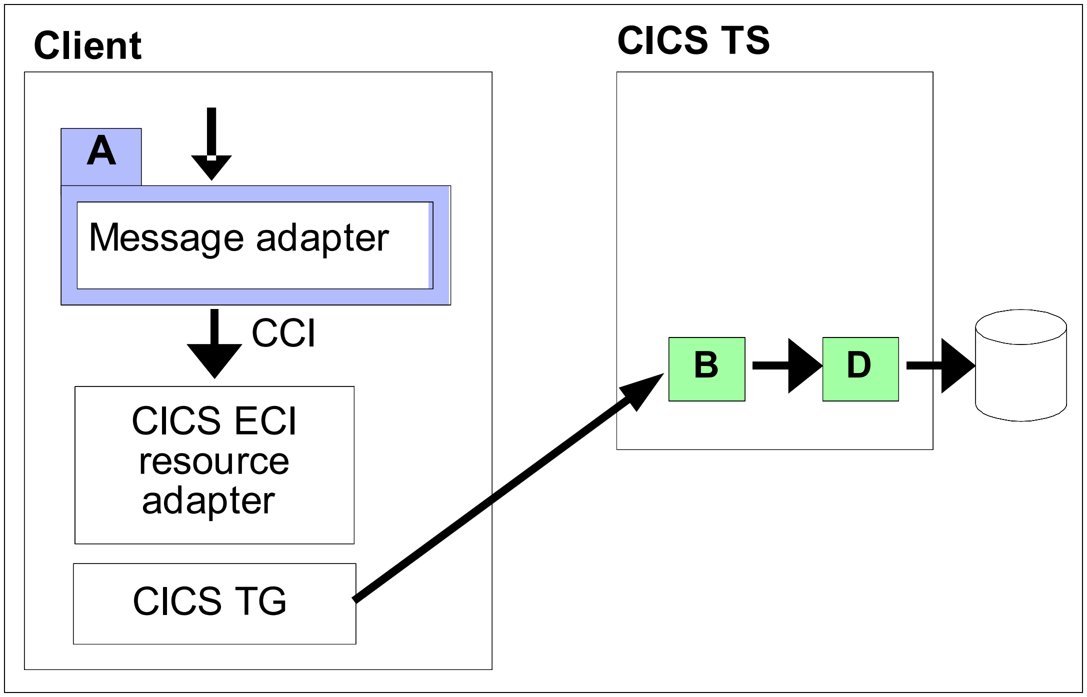


Figure 2-6 Connecting using JCA

A JEE application uses the CCI programming interface to invoke the CICS ECI resource adapter. The CICS TG ECI classes are packaged with the ECI resource adapter and are used to pass the application request to the CICS TG.

The JEE application can invoke the CICS business logic program (B) directly if no message transformation is required. In this case, Rational Application Developer can be used to create a Java bean to represent a COMMAREA formatted as COBOL types, with Java methods for getting and setting fields.

A message adapter in CICS is required only if the message is to be transformed. For example, the request is in XML and the CICS business logic program requires a COBOL record format. The length of the message is subject to the normal CICS COMMAREA message length limitation of 32 KB.

The CICS TG is the preferred implementation for JCA connectors to access all CICS servers from WebSphere Application Server, for applications that require a high-performing, secure, and scalable access option with tight integration to existing CICS applications. The CICS TG benefits from ease of installation and flexible configuration options, and requires minimal changes to CICS, and in most cases no changes to existing CICS applications. In addition, the CICS TG supports a range of non-Java clients, including C, C++, COBOL, and .NET.

The JCA is considered a medium coupling architecture, compared with the EJB architecture (high coupling) and the web services architecture (low coupling). The JCA can be used with a diverse range of supported environments and different deployment options. These are described in detail in 3.2, “CICS Transaction Gateway” on page 49.

2.4.3 HTTP and Atom feeds

Figure 2-7 shows how the HyperText Transmission Protocol (HTTP) can be used directly with CICS TS. CICS web support provides an HTTP listener and a message adapter program can be written using CICS WEB APIs.

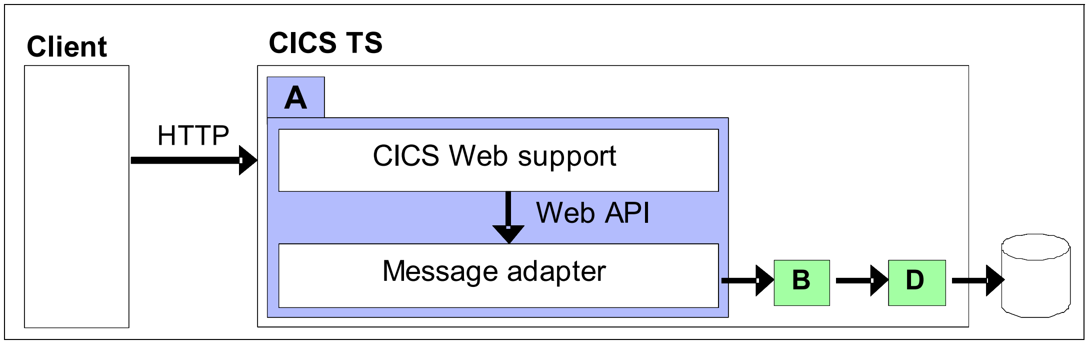


Figure 2-7 Connecting using CICS web support

CICS supports HTTP basic authentication for user ID identification or the more secure SSL encryption and authentication with client and server certificates. CICS web support sets up the transaction and security environment and calls the message adapter. The message adapter uses the CICS WEB APIs to extract the HTTP user data, which is typically formatted as an HTML form. The message adapter also has access to the SSL certificate and HTTP headers and socket information if required. The message adapter transforms this information and places it into containers or a COMMAREA and calls the business logic program.

Note: CICS TS currently supports the HTTP 1.0 and HTTP 1.1 specifications.

If the service requester is a web browser, the response message will typically be formatted as HTML. If the service requester is an application, the response message will normally be formatted as XML. The message adapter can use the CICS XMLTRANSFORM or DOCUMENT APIs to create HTML or XML documents. The response is returned to the client for display or processing.

HTTP is synchronous and stateless. However, if state management is required, CICS provides a utility for storing state data indexed by a state management token that the HTTP client can return on subsequent calls to retrieve the state.

CICS web support also allows a CICS application to initiate an HTTP request and to receive the response from an HTTP server program, thus providing bi-directional support for the HTTP protocol.

CICS web support also forms the basis for the CICS support of Atom feeds that adhere to the Atom Syndication Format and the Atom Publishing Protocol. CICS exposes resources such as VSAM files and temporary storage queues as Atom feeds without data access programs. For more information about CICS web support and Atom, refer to 3.4, “CICS web support” on page 63.

2.4.4 Messaging

WebSphere MQ allows you to easily exchange information across different platforms, integrating existing business applications in the process. WebSphere MQ assures reliable delivery of messages, dynamically distributes workload across available resources, and helps to make programs portable.

WebSphere MQ provides Java Message Service (JMS) APIs and native WebSphere MQ APIs for use by service requesters on a wide variety of platforms, with many options for routing and encrypting messages prior to arriving on WebSphere MQ for z/OS.

Figure 2-8 shows the WebSphere MQ trigger monitor program provided by CICS, which can be used to automatically start an appropriate message adapter program when messages arrive. The message adapter uses WebSphere MQ native APIs to receive the message, transform it if required, and call the business logic program. A reply message can be sent using the reply-to queue defined in the message. For efficiency, the message adapter program will usually continue to process messages on the inbound queue until it is empty.

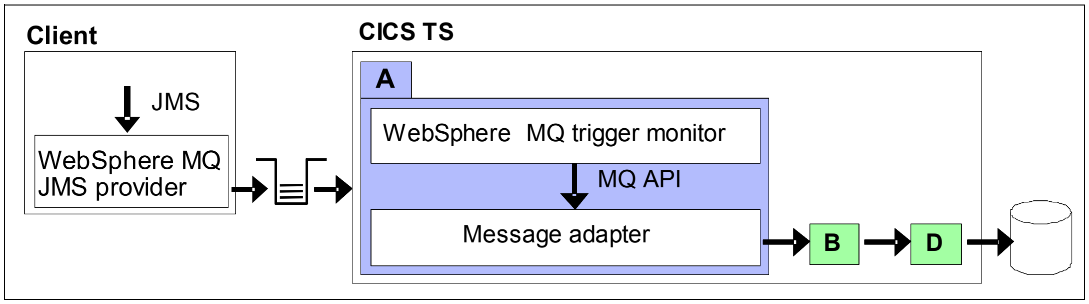


Figure 2-8 Connecting using WebSphere MQ

The WebSphere MQ DPL bridge for CICS provides an alternative option (Figure 2-9). This generic adapter passes a message from a named input queue to a business logic program through the COMMAREA. This is ideal in the situation where the service requester can format the message into a form acceptable by the business logic program.

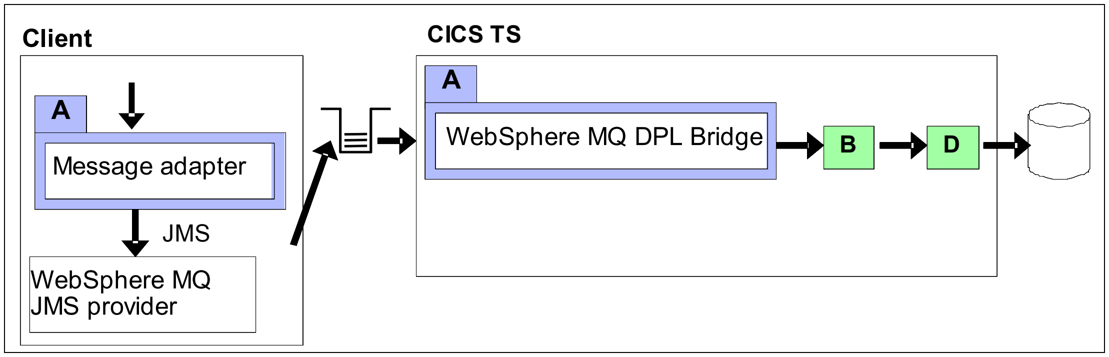


Figure 2-9 Connecting using the WebSphere MQ DPL bridge

When using the WebSphere MQ DPL bridge, the client application writes a structured message to the queue. This message must contain information in a predefined format that the monitoring transaction can use to decide how to handle the message. Several formats are possible, each starting with a block of data called an MQMD header. This field contains control information used by the monitoring transaction like the message format type, along with optional information, such as a reply-queue identifier and a user ID.

For more information about the use of WebSphere MQ with CICS, refer to 3.5, “WebSphere MQ” on page 68.

2.4.5 TCP/IP sockets

The TCP/IP Socket Interface for CICS (also known as *CICS sockets*) is part of z/OS Communications Server and supports peer-to-peer applications in which both ends of the connection are fully programmable (Figure 2-10). CICS sockets is most suitable when you are required to use a protocol not already supported by CICS TS.

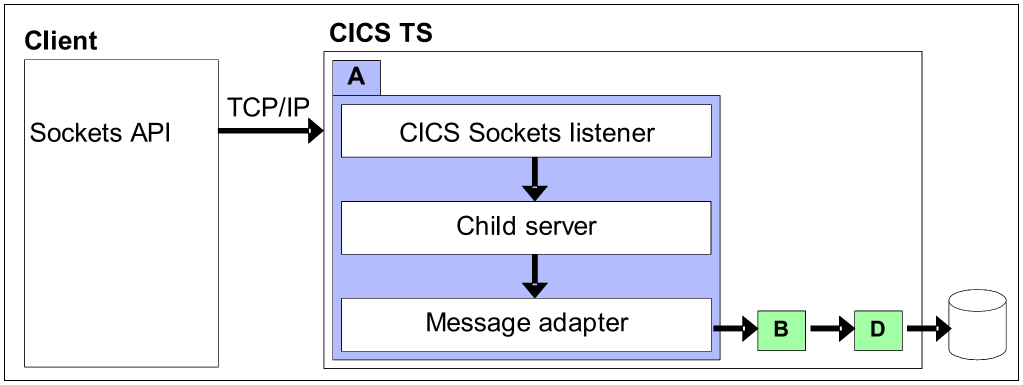


Figure 2-10 Connecting using CICS sockets

CICS sockets is configured and managed using CICS sockets transactions and configuration files rather than CICS systems management facilities.

CICS sockets provide an iterative listener and a concurrent listener, or you can write your own listener to meet your needs. The listener and child server use the CICS sockets APIs to open connections, send and receive data, and perform general communications control functions. The programs can be written in COBOL, PL/I, assembler language, or C. Client adapters can be written to create new outbound connections.

2.5 Conclusion

CICS provides a range of access methods to support modern connectivity architectures, such as web services and JEE, and other standard transport mechanisms. With the right external connectors and internal adapters, you can maximize the reuse of your existing mission-critical CICS assets. Table 2-1 compares the connection architectures and standard transport mechanisms discussed in this chapter.

Table 2-1 Common architectures and standard transport mechanism

Connectivity option	Required middleware	Main capabilities	Recommendation
Web services	CICS-only solution	► Inbound and outbound ► Low coupling ► Synchronous (HTTP) ► Asynchronous (WMQ) ► QoS based on transport type ► Support for some WS-* standards	Should be first consideration for service enabling CICS applications, particularly when you need to support multiple service requester types or need bi-directional support.
JCA with CICS TG	CICS TG JEE server (normally WebSphere Application Server)	► Inbound to CICS ► Medium coupling ► Synchronous ► High qualities of service (QoS)	Most appropriate solution when service requester is JEE component and when high QoS required (high availability, transactions, security).
JCA with WOLA	WebSphere Application Server for z/OS	► Inbound and outbound ► Tight coupling ► Synchronous ► High QoS	Particularly useful for JCA access to and from CICS, and for very high throughput and performance requirements.
CICS web support	CICS-only solution	► Inbound and outbound ► Medium coupling ► Synchronous ► Medium QoS	Use with web services, RESTful services, and Atom feeds, or when remote client/server only supports HTTP.
WebSphere MQ for z/OS	CICS-only solution	► Inbound and outbound ► Medium coupling ► Asynchronous, with almost-synchronous capabilities ► Assured delivery	Exploit WMQ for basic messaging and flowing web services.

Connectivity option	Required middleware	Main capabilities	Recommendation
CICS sockets	z/OS Communications Server	► Inbound and outbound ► Very tight coupling ► Synchronous ► Limited QoS	Use when remote client/server only supports TCP/IP sockets communication.

Both CICS Transaction Server and WebSphere Application Server are strategic middleware products that interoperate well using technologies, such as web services, to support end-to-end on demand systems. They exploit and complement z/OS qualities of service, such as high availability and scalability at a low cost per transaction, with a high level of security. In combination, WebSphere Application Server and CICS support almost any mission-critical SOA solution.

3

Technology overview

In this chapter, we introduce the CICS access technologies. We provide product information and an overview of the components and topologies.

These are the CICS access technologies that we discuss here and again in later chapters:

- CICS Web services
- CICS Transaction Gateway
- WOLA
- CICS web support
- WebSphere MQ
- CICS sockets

3.1 CICS Web services

Application programs running in CICS can participate in a heterogeneous web services environment as service requesters, service providers, or both. Figure 3-1 shows an outline of the web services support in CICS.

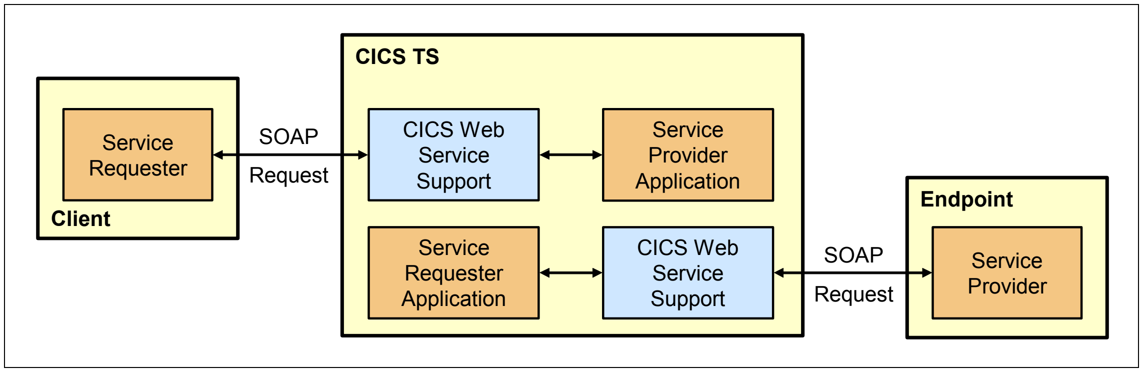


Figure 3-1 Web services in CICS

CICS support for web services conforms to open standards, including these standards:

- SOAP 1.1 and 1.2
- HTTP 1.1
- WSDL 1.1 and 2.0

CICS supports the most common type of communication between service requester and service provider: SOAP over HTTP.

CICS also receives and sends SOAP messages to WebSphere MQ (WMQ) using the WMQ transport, both in the role of service provider and service requester.

How does CICS support web services? Figure 3-2 shows an overview of how the components of the CICS Web services support fit together.

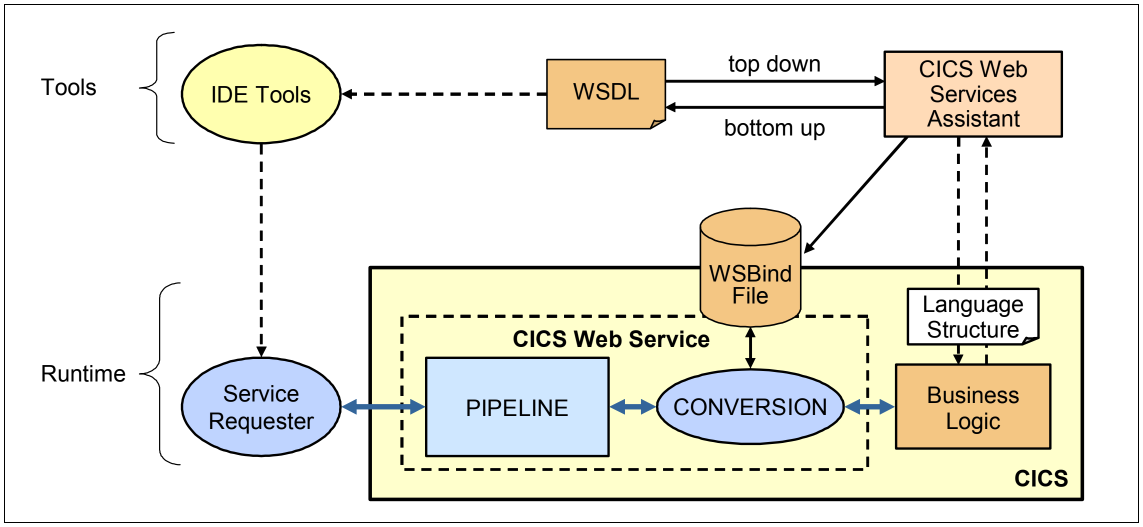


Figure 3-2 Overview of CICS Web services support

Figure 3-1 on page 38 makes a clear distinction between the tools and the runtime components of CICS Web service support. The tooling includes the IDE called Rational Developer for System z (RDz) and the CICS Web service assistant batch utilities. The runtime includes pipeline, message handlers, and CICS resource definitions. The section below provides more details on the tools and run time.

Tools

What tools are available to enable CICS programs as web services?

- CICS Web service assistant

  The CICS Web services assistant is a set of batch utilities that can help you transform existing CICS applications into web services and enable CICS applications to use web services provided by external providers.

  The assistant can create a WSDL document from a simple language structure like a COBOL copybook, or a language structure (copybook) from an existing WSDL document, and supports COBOL, C/C++, and PL/I. It also generates information used to enable automatic runtime conversion of the SOAP messages to containers and COMMAREAs, and vice versa.

  The assistant generates a WSBIND file. The WSBIND file is used by CICS to do message transformation (SOAP to application data and vice versa).

- Rational Developer for System z (RDz)

  RDz facilitates the development of both Java and z/OS-based applications. The XML services for the Enterprise (XSE) capability of RDz provides tools that let you adapt COBOL-based applications so that they can consume and produce XML messages. The Web services Enablement wizard is the XSE tool that supports the bottom-up approach for creating web services based on existing CICS COBOL programs.

Both the tools described above can be used to build web services in various development styles. You can use the tools to build a web service in top-down style or bottom-up style or meet in the middle:

- Top-down style

  This approach is usually the starting point when we have an existing WSDL document for a web service and we want to either implement or invoke the web service within CICS. We can use either CICS Web service assistant or RDz to do this.

- Bottom-up style

  This approach is usually the starting point when we have an existing CICS application that is already in production and has either a COMMAREA or channel-based interface. We now want to expose this application to remote client applications using CICS Web services support.

- Meet in the middle

  This is a hybrid technique and often involves the use of a wrapper program that maps between the data format generated by the CICS Web service assistant (or RDz) and the desired data format used by the existing application. This is used in complicated scenarios where COMMAREA fields or language are not supported by the CICS Web service assistant.

3.1.1 Components for CICS Web service

In this section we discuss components for CICS Web service.

Base components

In addition to the TCP/IP Listener, Secure Sockets Layer (SSL) support, and the HTTP handler (as described in 3.4.3, “Components for CICS web support” on page 65), CICS also provides the following base components for CICS Web service:

- WMQ Listener: The WMQ listener receives WMQ messages and invokes pipeline processing.
- SOAP protocol handler: The SOAP protocol handler receives SOAP messages and invokes pipeline processing.
- XML data mapper: The XML data mapper maps SOAP message to COMMAREA/containers.

Resource components for web service requests

What are the resource definitions that are required to support web services?

The following CICS resources are configured by a systems programmer to access CICS services over HTTP:

- TCPIPSERVICE

  A TCPIPSERVICE definition is required in a service provider that uses the HTTP or HTTPS as transport. It contains information about the port on which inbound requests are received, and whether any transport-based security mechanisms will be applied by CICS.

- URIMAP

  A URI mapping or URIMAP resource definition matches the URIs of web service requests. The URIMAP associates a URI for the request with a PIPELINE and WEBSERVICE resource that specifies the processing to be performed. You can use a URIMAP to specify:

  – The name of the transaction that CICS uses for running the pipeline alias transaction (default is CPIH)

  – The user ID under which the alias transaction runs

- PIPELINE

  A PIPELINE resource definition provides information about the message handlers that will act on a service request and on the response. The information about the message handlers is supplied indirectly. The PIPELINE definition specifies the name of an HFS file, called the pipeline configuration file, which contains an XML description of the message handlers and their configuration. There are two kinds of pipeline configuration files:

  – One describes the configuration of a service provider pipeline.
  – The other describes the configuration of a service requester pipeline.

- WEBSERVICE

  A WEBSERVICE resource defines the aspects of the runtime environment for a CICS application program deployed in a web services setting. The CICS Web service assistant is used to generate the mapping between the application data structure and SOAP messages. The mapping is stored in the WSbind file. Three objects define the execution environment that allows a CICS application program to operate as a web service provider or a web service requester (Figure 3-3 on page 43):

  – WEBSERVICE resource: The web service description. This is used to reference the WSDL file, PIPELINE, and WSbind file.

  – WSbind file: The web service binding file.

  – PIPELINE resource definition.

Note: What is a web service binding file?

A web service binding file contains abstract representations of the input and output messages used by the service. When a service provider or service requester application executes, CICS needs information about how the content of the messages maps to the data structures used by the application. This information is held in a web service binding file.

3.1.2 CICS resource relationships

Figure 3-3 shows the relationships between CICS Web services definitions.

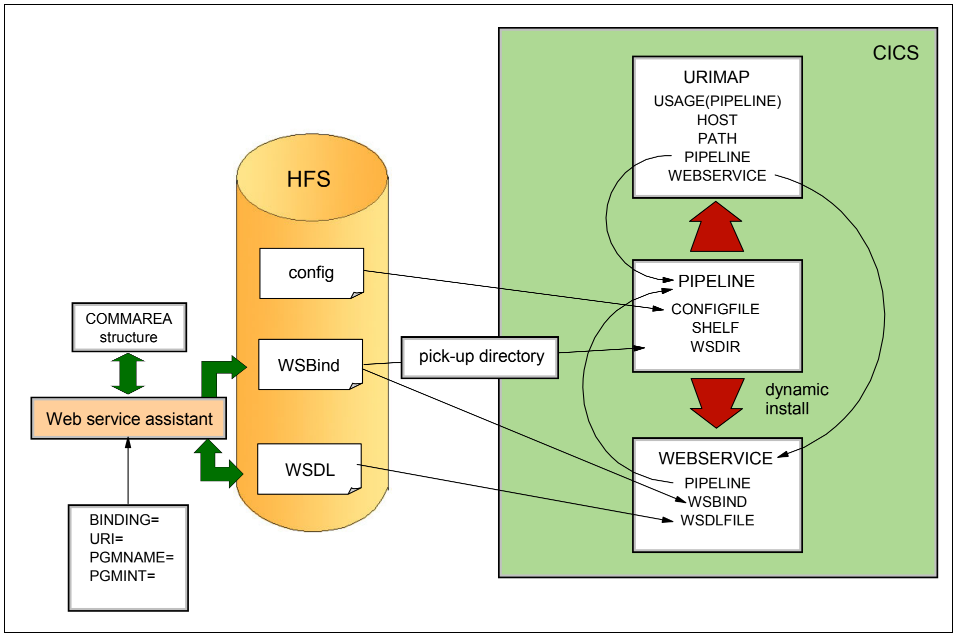


Figure 3-3 CICS Web services resource interrelationships

These are the steps performed when CICS is a service provider:

1. When a web service request is received, CICS searches for a matching URIMAP resource with its USAGE attribute set to PIPELINE and its PATH attribute set to the URI found in the incoming request.
2. If a matching URIMAP definition is found, the PIPELINE and WEBSER VICE definitions from the PIPELINE and WEBSERVICE attributes of the URIMAP definition are used.
3. The TRANSACTION attribute of the URIMAP definition determines the name of the transaction that should be attached to process the pipeline.

Note: CICS support for web service standards does these things:

- It ensures maximum interoperability with other web services implementations by conforming with the Web Services Interoperability Organization (WS-I) Basic Profile 1.1 and WS-I Simple SOAP Binding Profile1.0.
- It supports the WS-Atomic Transaction and WS-Coordination specifications. By default, web service requests are stateless. The above specifications add a layer for transactionality.
- It supports the WS-Security specification. WS-Security is a large specification that deals with the many aspects of security related to web services. It covers everything from passing a user ID, single-field encryption, message encryption, signing, and so on. For more information about WS-Security, see Chapter 6, “Security” on page 133.
- It conditionally complies with WS-Trust, and support is subject to restrictions. WS-Trust (one of the six sub-specifications of WS-Security) allows you to support more forms of identification. CICS, for example, only supports username tokens and X509 certificates. For more information about WS-Trust, see Chapter 6, “Security” on page 133.

3.1.3 CICS as a service provider application

In this section we first discuss how to prepare for running a CICS application as a service provider. Then we discuss how CICS processes the incoming service request.

An existing COMMAREA-based application can be exposed as a service provider, normally without any application changes. When CICS is in the role of service provider, it must perform the following operations:

1. Receive the request from the service requester.
2. Examine the request and extract the contents that are relevant to the target application program.
3. Invoke the application program, passing data extracted from the request.
4. Construct a response (when the application program returns control) using data returned by the application program.
5. Send a response to the service requester.

3.1.4 CICS as a service requester application

When CICS is a service requester, an application program sends a request, which is passed through a pipeline to a target service provider. The response from the service provider is returned to the application program through the same pipeline. In this section we discuss how to prepare for running a CICS application as a service requester. Then we discuss how CICS processes the outbound service request.

When CICS is in the role of service requester, it must perform the following operations:

1. Build a request using data provided by the application program.
2. Send the request to the service provider.
3. Receive a response from the service provider.
4. Examine the response and extract the contents that are relevant to the original application program.
5. Return control to the application program.

3.1.5 Web services using WebSphere MQ as transport

CICS can receive and send SOAP messages to WebSphere MQ using the WebSphere MQ transport, both in the role of service provider and service requester. As a service provider, CICS uses WebSphere MQ triggering to process SOAP messages from an application queue. Triggering works by using an initiation queue and local queues. A local (request) queue definition includes the following information:

- The criteria for when a trigger message is generated. For example, when the first message arrives on the local queue, or for every message that arrives on the local queue. For CICS SOAP processing, specify that triggering occurs when the first message arrives on the local queue.
- The local queue definition can also specify that trigger data is passed to the target application, and in the case of CICS SOAP processing (transaction CPIL), this specifies the default target URL to be used if this is not passed with the inbound message.
- The process name that identifies the process definition. The process definition describes how the message is processed. In the case of CICS SOAP processing, specify the CPIL transaction.
- The name of the initiation queue that the trigger message should be sent to.

When a message arrives on the local queue, the Queue Manager generates and sends a trigger message to the specified initiation queue. The trigger message includes the information from the process definition. The trigger monitor retrieves the trigger message from the initiation queue and schedules the CPIL transaction to start processing the messages on the local queue.

As a service requester, on outbound requests you can specify that the responses for the target web service are returned on a particular reply queue. In both cases, CICS and WebSphere MQ require configuration to define the required resources and queues. Figure 3-4 shows how WebSphere MQ acts as transport for CICS service requester and service provider applications.

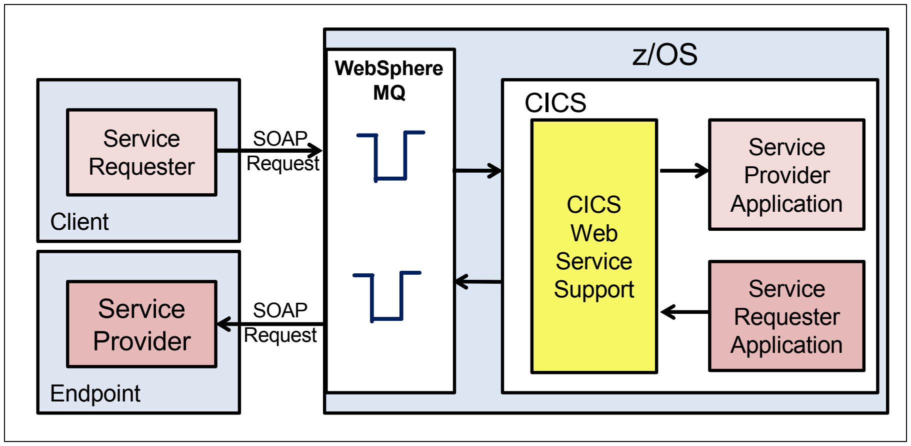


Figure 3-4 Example of WebSphere MQ transport flow

Figure 3-4 shows how WebSphere MQ acts as a transport for web service.

3.1.6 Support for MTOM

In standard SOAP messages, binary objects are base64-encoded and included in the message body, which increases their size by 33%. For very large binary objects, the larger payload can significantly impact transmission time. CICS SOAP pipelines can support the Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) specifications. These specifications define a mechanism for sending and receiving binary data using SOAP, without incurring the overhead of base64 encoding. CICS supports and controls the handling of MTOM messages in both web service provider and requester pipelines using an MTOM handler program and XOP processing.

3.1.7 Java web services using Axis2

Axis2 is a Java-based implementation of a web services SOAP engine that supports a number of the web services specifications. Axis2 is provided with CICS TS V4.2 to process web services in a Java environment.

You can optionally use the Axis2 Java-based SOAP engine to process web service requests in service provider and service requester pipelines. Because Axis2 uses Java, the SOAP processing is eligible for off loading to the IBM System z Application Assist Processor (zAAP). You can opt to use Axis2 by adding a Java SOAP handler to your pipeline configuration file and creating a JVM server to handle the Axis2 processing.

Enabling Axis2 does not require regenerating the binding files for any existing web services that use the pipeline. When CICS is a service provider, the Java-based handler uses Axis2 to parse the SOAP envelope for a request message. You can use header processing programs to process any SOAP headers associated with the SOAP message. Axis2also constructs the SOAP response message. Figure 3-5 shows this process.

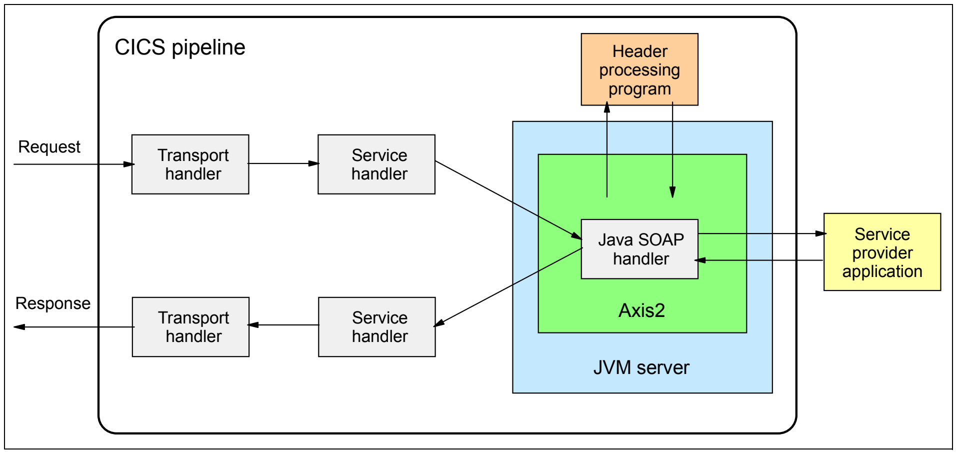


Figure 3-5 Axis2 processing when CICS is a service provider

Important: See Chapter 7, “Transactional scope” on page 185, for more information about scalability.

3.1.8 Java web service topology using Axis2

In this section we discuss Java web service topology using Axis2.

Topology 1

In this topology, a Java application has been written using the Java Architecture for XML Binding (JAXB) and the Java API for XML Web Services (JAX-WS) libraries to generate and parse the XML. The Java application can now run in Axis2 in the same JVM server as the SOAP pipeline processing (Figure 3-6). The topology in Figure 3-6 is an example of a Java application that is a web service provider and is processed by the Axis2 SOAP engine in a JVM server. JCICS is the CICS Java class library that wrappers EXEC CICS commands and provides to CICS resources such as VSAM files and TSQs.

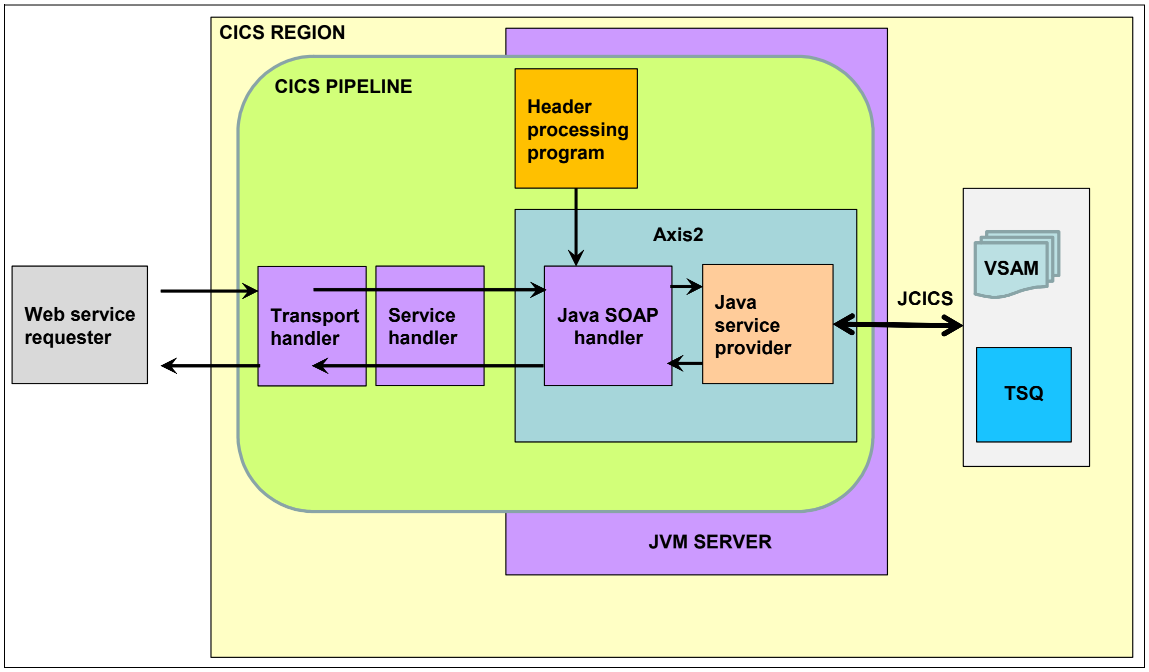


Figure 3-6 Web service with Java service provider and Axis2

Topology 2

The topology shown in Figure 3-7 is an example of a COBOL application that is a web service provider. The request is processed in a pipeline that is configured to support Java. The SOAP handler is a Java program that is processed by Axis2 and run in a JVM server.

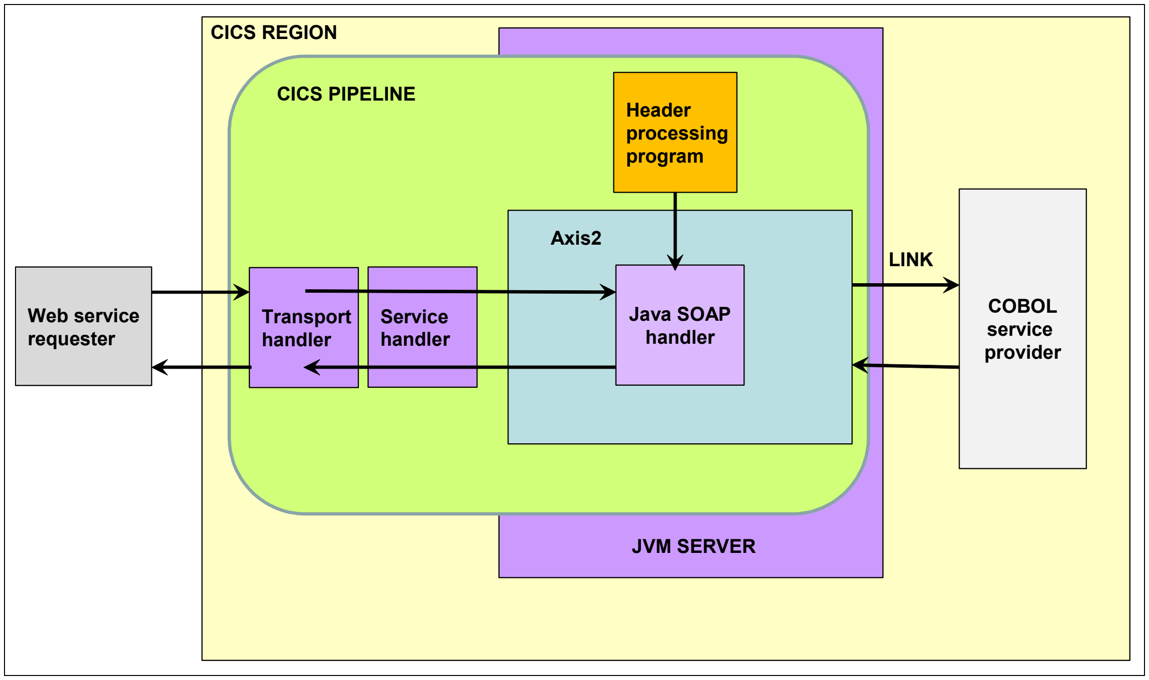


Figure 3-7 Web service with COBOL service provider and Axis2

Refer to Chapter 7, "Transactional scope" on page 185, for more information about scalability of the above topologies.

3.2 CICS Transaction Gateway

The CICS Transaction Gateway (CICS TG) is a set of client and server software components that allow a remote client application to invoke services in a CICS region. The client application can be either a Java application or a non-Java application using C, C++, workstation COBOL, or .NET interfaces (depending on the platform used).

When a Java application is used, then the application can be any type of client (such as a servlet or an enterprise bean). In the JEE environment, the application is typically a servlet or enterprise bean that is deployed into a JEE application server, such as WebSphere Application Server. Figure 3-8 shows how CICS assets can be accessed from multiple environments with CICS TG.

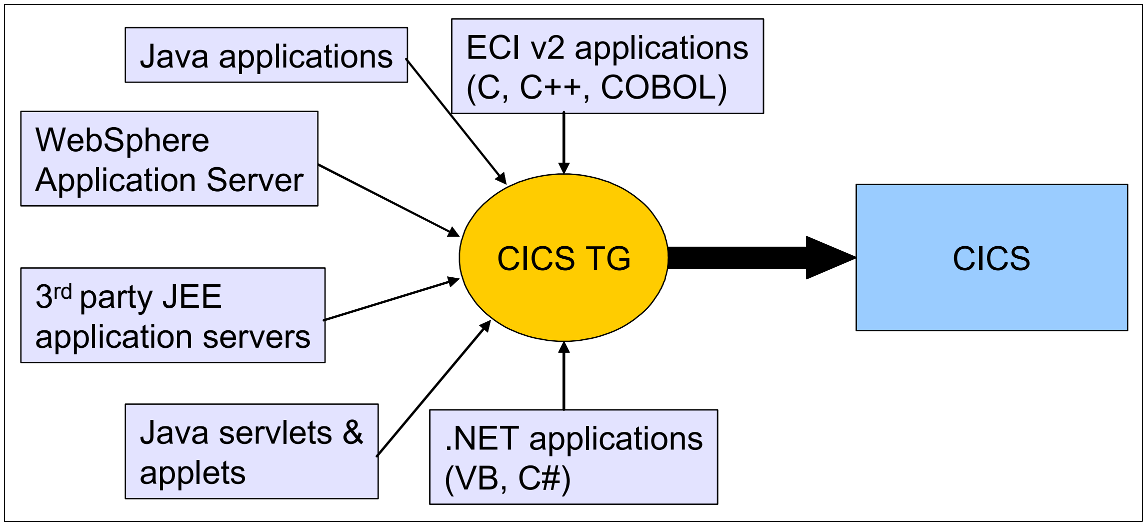

Java applications
ECI v2 applications (C, C++, COBOL)
WebSphere Application Server
CICS TG
CICS
3rd party JEE application servers
Java servlets & applets
.NET applications (VB, C#)

Figure 3-8 Extending CICS assets with CICS TG

3.2.1 CICS TG products

With CICS TG there are three distinct CICS TG products:

- CICS TG for Multiplatforms
- CICS TG Desktop Edition
- CICS TG for z/OS

3.2.2 CICS TG for Multiplatforms

CICS TG for Multiplatforms is supported on the following range of operating systems and platforms and is designed to support connectivity to all in-service CICS servers:

- Linux on System z
- Linux on Intel
- Linux on POWER®
- AIX®
- HP-UX
- Oracle Solaris
- Microsoft Windows

CICS TG for Multiplatforms comprises the following main runtime components:

- The Gateway daemon, which listens for incoming work and manages the threads and connections necessary to ensure good performance. It provides connectivity to CICS when using the IPIC protocol.
- The Client daemon, which provides the communication to CICS servers.
- A Java class library or JCA resource adapter, which is deployed into the client runtime environment. When used in a JCA environment, the resource adapter is deployed into the JEE application server.
- A ECI v2 client library for use by remote C or .NET clients.

A Java client program can connect to a remote Gateway daemon using the TCP or SSL protocols. The Client daemon then provides the transport drivers to connect to the CICS server.

3.2.3 CICS TG Desktop Edition

CICS TG Desktop Edition provides all the capabilities of CICS TG that are described in the above section. It is licensed and limited to single user access. It replaces the earlier product CICS Universal Client, but without JCA support.

3.2.4 CICS TG for z/OS

CICS TG for z/OS is supported on z/OS and supports connectivity to CICS TS for z/OS. CICS TG for z/OS uses EXCI or IPIC connections provided by CICS TS to communicate with CICS (Figure 3-9). ECIv2 provides support for non-Java-based applications.

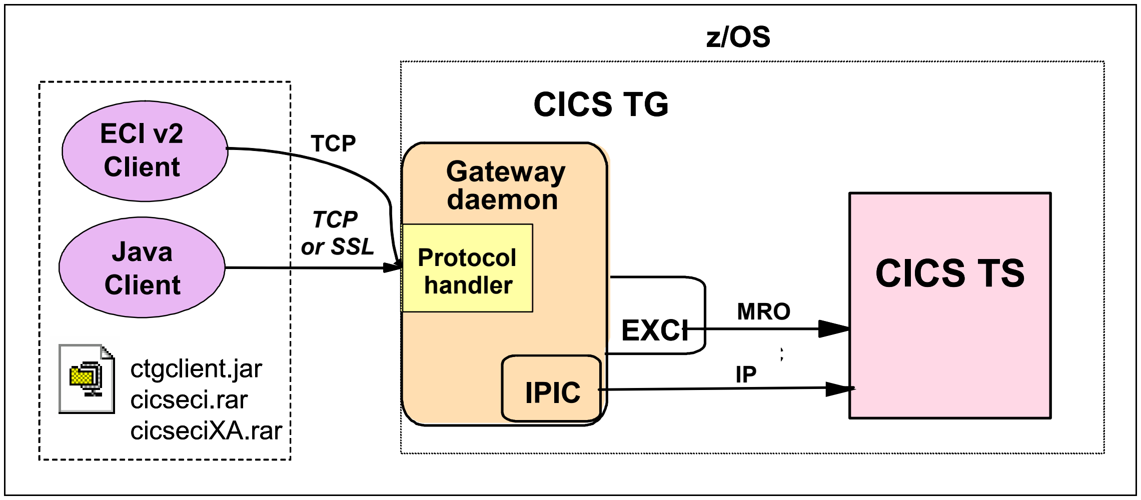


Figure 3-9 Components of CICS TG for z/OS

Figure 3-9 depicts how Java clients and ECIV2 clients can connect to CICS using the CICS TG on z/OS. CICS TG can be configured to use EXCI or IPIC to communicate with CICS.

3.2.5 CICS TG for z/OS modes of operation

There are two principle modes of operation for the CICS TG:

- Remote
- Local

Remote mode of operation

The remote mode of operation uses the Gateway daemon as a long-running task that listens on specified ports for incoming ECI requests and then forwards them to the CICS server via the EXCI or the IPIC protocol. The Gateway daemon runs in its own address space and provides connection and thread management.

Local mode of operation

This is only available for Java clients. If the CICS TG is to be used on the same machine as WebSphere Application Server, it might be more efficient to use the CICS TG classes within WebSphere Application Server to provide the gateway functionality. This mode of operation allows WebSphere Application Server to manage the connections and threads and reduces the communications overhead. This configuration is known as the local mode of operation.

Note: To read more about the CICS TG local mode of operation, see this URL:

`http://www.redbooks.ibm.com/abstracts/sg247161.html`

CICS TG for z/OS has many advantages in the area of systems management, usability, and performance:

- XA transaction support enables CICS Transaction Server (CICS TS) for z/OS to participate in a global two-phase commit transaction that is initiated in a distributed JEE application server, such as WebSphere Application Server.
- Management of the Gateway daemon from SDSF provides better system administration capabilities.
- Better security with identity propagation, identity assertion, and SSL directly into z/OS.

3.2.6 CICS TG application programming interfaces

CICS TG for Multiplatforms provides the following programming interfaces for accessing CICS applications:

- External Call Interface (ECI)
- External Presentation Interface (EPI)
- External Security Interface (ESI)

Note: The CICS TG on z/OS supports only the ECI and ESI interface.

External Call Interface

The ECI is used for calling COMMAREA or channel-based CICS programs. The COMMAREA or channel is the buffer that is used for passing the data between the client and the CICS server. CICS sees the client request as a distributed program link (DPL) request.

The ECI enables a user application to call a CICS program synchronously or asynchronously. It enables the design of new applications to be optimized for client-server operation, with the business logic on the server and the presentation logic on the client.

An ECI request can be invoked from a Java application using a variety of interfaces:

- The ECIRequest class that is provided by the CICS TG base classes

  This interface provides a simple procedural type interface to the ECI. It is supported in any Java environment (such as a stand-alone application) and provides similar capabilities to the JCA. However, it does not provide the same qualities of service (such as XA transaction support).

- The Common Client Interface (CCI) that is provided by the CICS ECI resource adapters (cicseci.rar)

  These classes define a standard architecture for connecting the Java 2 Platform Enterprise Edition (JEE) platform to a heterogeneous EIS, such as CICS. Java applications interact with resource adapters using the Common Client Interface (CCI), which is a common framework of classes extended by each resource adapter to allow communication with a specific EIS. JCA provides a high quality of service, such as managed security, identity propagation, and XA (2-phase commit).

The ECIv2 API enables a C client application to communicate with a CICS TG running on a remote machine in much the same manner as the Java APIs. An application developer using this API is able to call COMMAREA or channel-based CICS applications. Applications can make use of the transaction support in CICS either by having each program run in its own transaction or by multiple calls within an extended logical unit of work (LUW). Applications written using this API can be developed for the Windows, Linux, or UNIX platforms that are supported by the CICS TG. This function is similar to that provided by the existing Client APIs in the CICS TG. However, an application written using ECIv2 is not required to run on the same machine as a CICS TG installation.

An ECI request can use the following connections:

- EXCI
- IPIC

What is the benefit of IPIC versus EXCI? IPIC provides channel/container support, sysplex-wide XA support, and identity propagation.

External Presentation Interface

The EPI is used for invoking 3270-based transactions. A terminal is installed in CICS, and CICS sees the request as running on a remote terminal controlled by the CICS TG. This interface is not supported on z/OS.

External Security Interface

The ESI is used for verifying and changing the user ID and password information held in the CICS external security manager (ESM), such as RACF®.

ESI calls to CICS can be made from Java, .NET, or C clients. It provides two basic functions:

Verify Password	This allows a client application to verify a password for a given user ID.
Change Password	This allows a client application to change the password for a given user ID.

3.2.7 CICS TG and the JCA

The CICS TG is a JEE connector for CICS TS, and in conjunction with IBM WebSphere Application Server provides a high-performing, secure, scalable, and tightly integrated access method in CICS.

The JCA system-level contracts between a JEE application server, such as WebSphere Application Server and a resource adapter, determine the scope of the JCA managed environment. The standard contracts include a connection-management contract, transaction-management contract, and security-management contract. These contracts provide the mechanisms by which the management of connections, security, and transactions are performed:

- The connection-management contract enables the application server to pool and re-use connections into CICS, enabling a more scalable and efficient environment that can support a large number of concurrent accesses to a CICS region.
- The transaction-management contract defines the scope of transactional integration between a JEE application deployed in WebSphere Application Server and a CICS program.
- The security-management contract defines how security context information is passed between the application server and CICS.

3.2.8 Using the CICS ECI resource adapter with different topologies

The JCA system contracts are the key to the qualities of service provided by WebSphere Application Server and the CICS ECI resource adapter. However, the qualities of service vary depending on the topology in use. The three most common topologies are shown in Figure 3-10:

Topology 1	WebSphere Application Server and the CICS TG are both deployed on a distributed (non-System z) platform.
Topology 2	WebSphere Application Server is deployed on a distributed platform and the CICS TG is deployed on a z/OS system.
Topology 3	Both WebSphere Application Server and the CICS TG are deployed on System z.

We discuss topology 2 in more detail in Chapter 5, "Application interfaces" on page 89, Chapter 6, "Security" on page 133, Chapter 7, "Transactional scope" on page 185, Chapter 8, "High availability and scalability" on page 219, and Chapter 10, "CICS TG for z/OS scenario" on page 275.

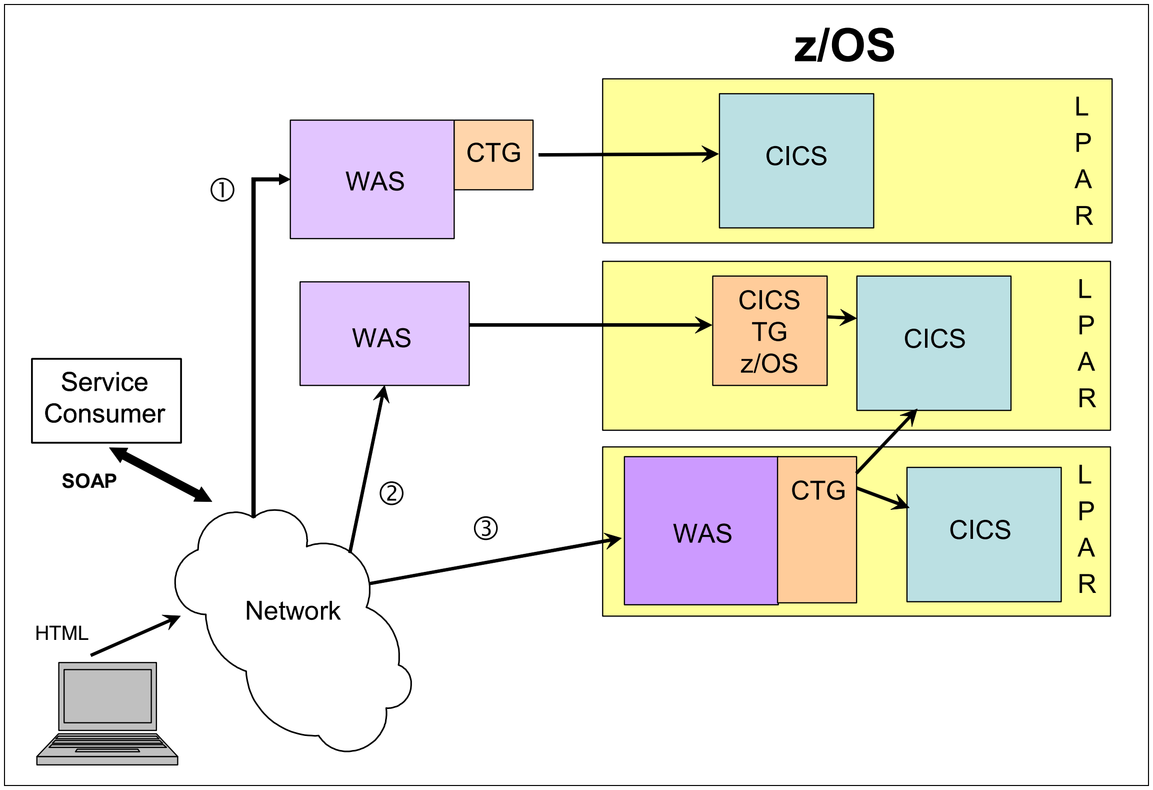


Figure 3-10 Common topologies for using CICS TG with WebSphere Application Server

Remote Gateway daemon on z/OS

In topology 2, where WebSphere Application Server is deployed on one of the distributed platforms, it is possible to access CICS through a Gateway daemon running on z/OS (Figure 3-11).

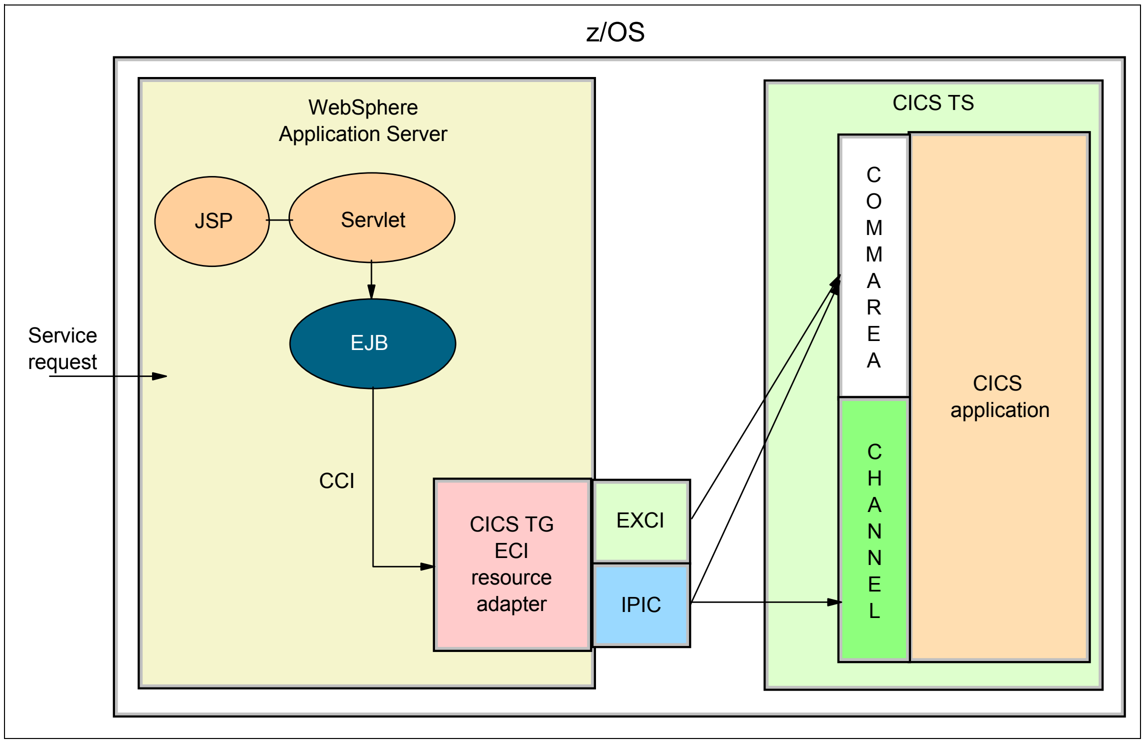


Figure 3-11 CICS TG topology 2

In this configuration, the protocol used is one of the remote protocols (TCP, HTTP, SSL, or HTTPS). The communication from the CICS Transaction Gateway on z/OS to the CICS server utilizes EXCI or IPIC.

This configuration is widely used today for the following reasons:

- Topology 1 only supports native TCP/IP connections into System z systems for CICS TS V2 onwards, whereas topology 2 provides simple TCP/IP access to any release of CICS.
- Topology 2 provides advanced security with identity assertion and identity propagation. See Chapter 6, “Security” on page 133, for details.
- Topology 2 enables integration with z/OS IP workload-management functions, including Sysplex Distributor and TCP/IP port sharing for better HA. See Chapter 8, “High availability and scalability” on page 219, for details.

- Deploying the CICS TG on z/OS can leverage the existing CICS systems-management skills within the enterprise for better system administration.
- Topology 2 provides XA transaction support to enable resources to participate in two-phase commit transactions. See Chapter 7, “Transactional scope” on page 185, for details

3.3 WOLA

WebSphere Optimized Local Adapters (WOLA) is a functional component of WebSphere Application Server for z/OS that provides an efficient cross-memory mechanism for calls both inbound and outbound to and from WebSphere Application Server for z/OS. It can communicate with external address spaces, which include CICS, batch, IMS, USS, and ALCS. Because it avoids the overhead of other communication mechanisms, it is capable of high-volume exchange of messages. Figure 3-12 shows the topology of WOLA and CICS.

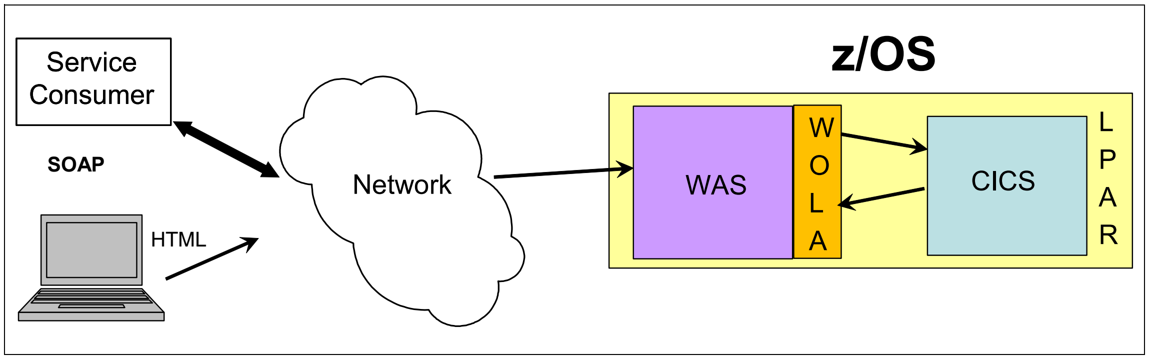


Figure 3-12 WOLA and CICS topology

3.3.1 What is WOLA

WOLA is a set of runtime components and APIs provided with WAS z/OS V7 and later that enable procedural style calls to and from other sub-systems on z/OS. It provides an set of APIs for sending data between different z/OS environments. WOLA provides interfaces to connect in and out of WebSphere Application Server for z/OS (bi-directional connectivity). WOLA has support for routing requests between different WAS servers with the development mode support. For inbound connections to CICS from WAS, JCA resource adapter hides most of the WOLA implementation details, which makes Java programming easy.

3.3.2 The benefit of WOLA

WOLA started out as a way to allow program access into WebSphere Application server for high transaction rate batch programs. The benefits of WOLA are:

- Integrated with WAS and thus cheap.
- Low pathlength and thus performs well.
- Bidirectional.
- For inbound connections to CICS from WAS, the JCA resource adapter (ola.rar) hides most of the WOLA implementation details, which makes Java programming easy.

A standard JCA resource adapter is used for deployment into WebSphere Application Server for z/OS so that Java programs can have a standard Common Client Interface (CCI) to interact with. The CCI interface is not the same one that is used by CICS TG.

3.3.3 CICS and WOLA

Figure 3-13 shows a high-level overview of the WOLA support in CICS. This section describes the WOLA CICS Link Server, but the WOLA APIs to accept work from WAS directly into a CICS program can be used as well. The Host Service and Receive Request WOLA APIs can be coded directly into a CICS program and can become the target of work requests from WAS.

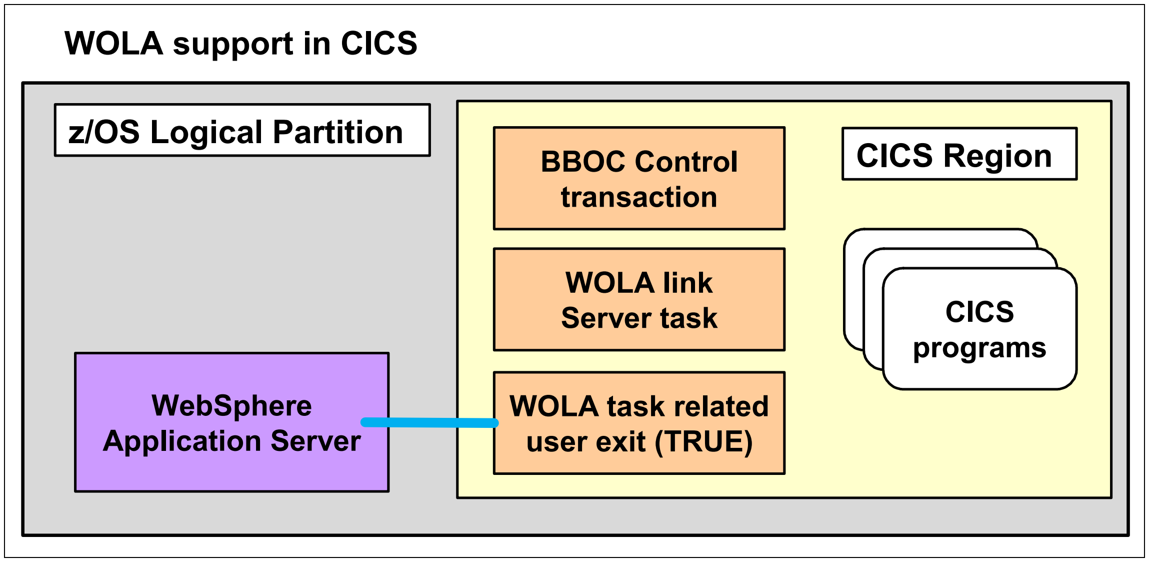


Figure 3-13 WOLA support in CICS

These are the three components inside CICS that provide WOLA support:

- The Task Related User Exit (TRUE)

  This is the low-level heart of the WOLA support in CICS. It provides the low-level module support for access to the local communication function of WAS z/OS. CICS cannot use WOLA without it, and this has to be installed in any CICS region that wants to use WOLA. The TRUE adapter is designed to run in a CICS region as a resource manager. In CICS, the TRUE is the primary vehicle used by resource providers. TRUE support provides the boundary between the CICS application threads and the external resource manager threads. Currently, DB2, WebSphere MQ, and TCPIP sockets execute in CICS using the TRUE support.

- The Link Server Task

  The link server task hides the CICS programs from the specifics of WOLA. It serves as a kind of *WOLA handler* from WAS to CICS. It accepts the request, then turns and uses EXEC CICS LINK (DPL) of the named target CICS program passing either COMMAREA or a single named container. The Link Server Task does not come into play for calls from CICS to WAS. For CICS to WAS calls, some custom coding needs to be done with the WOLA API.

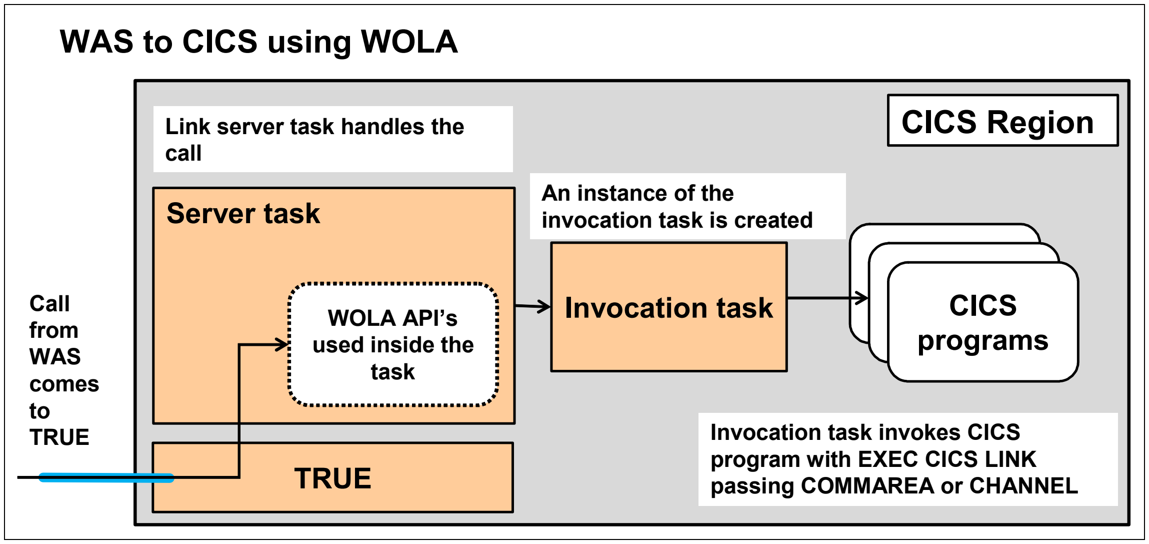
- BBOC 3270 control transaction

  This is a utility provided to make the management easier and is very handy. The BBOC control transaction provides a convenient way to start and stop the TRUE and the Link Server Task and pass in parameters to modify the behavior of the environment.

3.3.4 How calls to CICS work with WOLA

Any external address spaces (such as CICS) that use the supplied interfaces must first register with WOLA. A Java program from WebSphere Application server on z/OS wanting to initiate an optimized local adapter (OLA) call outbound can be implemented as either a servlet or EJB. The Java program uses the supplied JCA resource adapter (ola.rar) file to make the call to CICS.

In CICS, you can use the supplied link server task (BBO$) to act as the receiving agent on behalf of existing CICS program assets. BBO$ then issues an EXEC CICS LINK of the program named. No changes to the existing CICS program are necessary provided that it supports either COMMAREA or channel. WOLA supports only one input and one output container, and has a fixed channel name, so it is much more restrictive than traditional CICS channel/container support. Figure 3-14 shows how the call from WebSphere Application Server to CICS works.



WAS to CICS using WOLA
CICS Region
Link server task handles the call
Server task
WOLA API's used inside the task
An instance of the invocation task is created
Invocation task
CICS programs
Call from WAS comes to TRUE
TRUE
Invocation task invokes CICS program with EXEC CICS LINK passing COMMAREA or CHANNEL

Figure 3-14 Inbound call to CICS

3.3.5 How calls from CICS work with WOLA

This involves some degree of coding to the APIs. The link server task does not assist in calls outbound from CICS to WebSphere Application Server. There are two approaches for coding the APIs (Figure 3-15 on page 62):

- Embed the WOLA API processing in the CICS program itself. This is easy to do if you are writing a new application, or can easily modify an existing one, but it is a bit more problematic for existing programs where the mandate is to leave it untouched for modifications.
- The second option is to write a kind of custom WOLA bridge program that can be linked by your application programs. This becomes a kind of *WOLA Service* to existing CICS programs to utilize as your needs require.

Figure 3-15 provides an example of outbound call from CICS to WAS using WOLA.

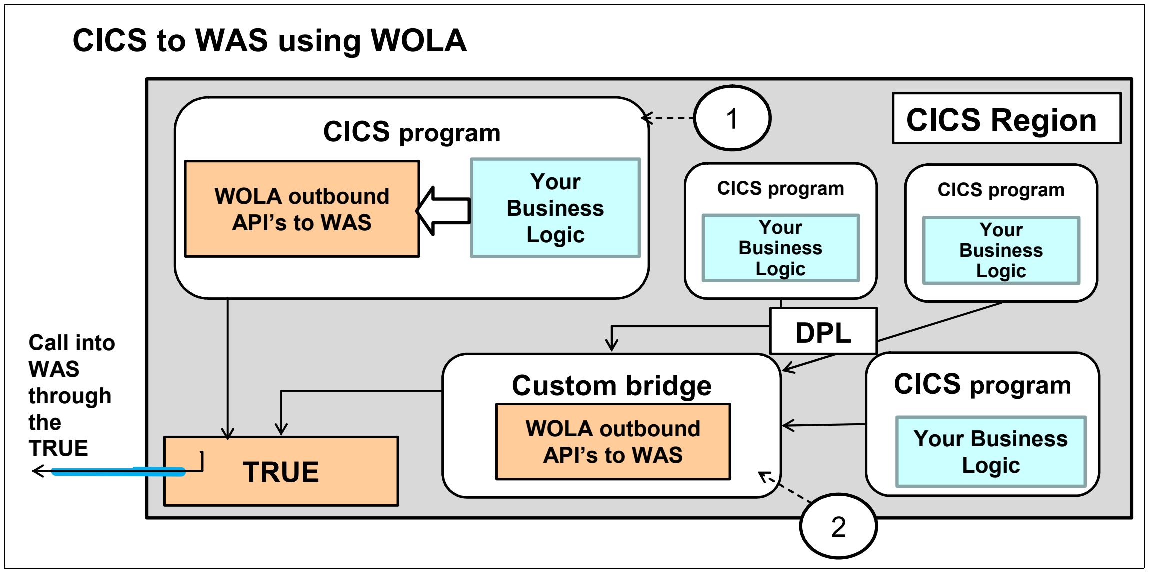


Figure 3-15 Outbound call from CICS

3.4 CICS web support

CICS Web support is a set of resources supplied with CICS TS for z/OS that enables a CICS region to act as an HTTP server and as an HTTP client. This allows CICS applications to be invoked by and reply to HTTP requests. Figure 3-16 shows an overview of CICS Web support.

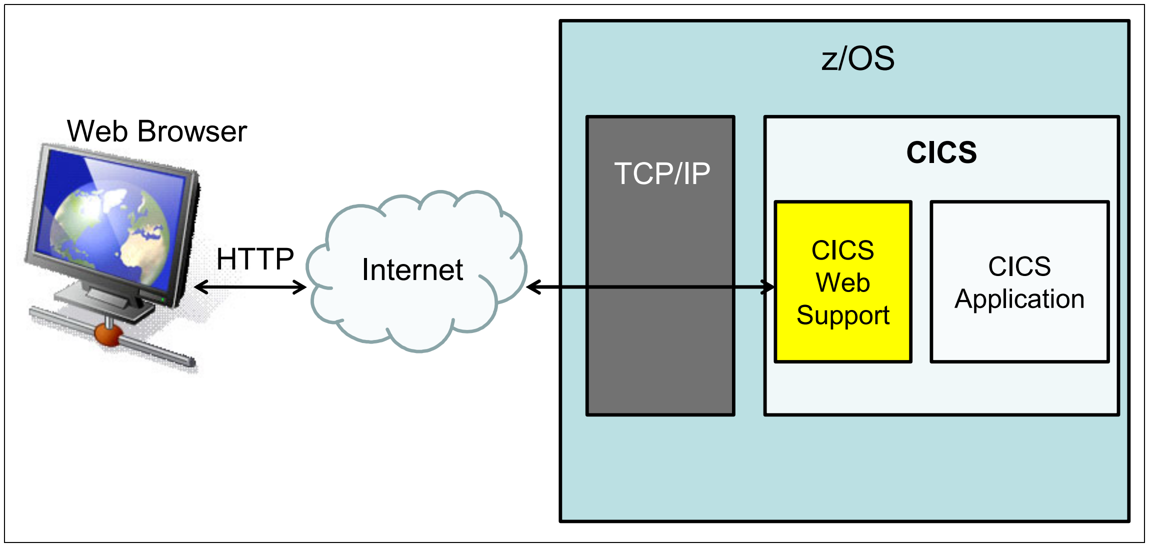


Figure 3-16 CICS Web support overview

3.4.1 CICS as an HTTP server

When CICS is an HTTP server, a web client can send an HTTP request to CICS and receive a response. The response can be a static response created by CICS from a document template or static file, or an application-generated response created dynamically by a user application program.

The actions of CICS as an HTTP server are controlled by TCPIPSERVICE definitions and URIMAP definitions, which are used to configure CICS web support and instruct CICS on how to process requests and responses.

Processing flow for CICS as an HTTP server

Figure 3-17 shows the process flow of CICS as an HTTP server:

1. The user invokes a CICS via a URL.
2. The long-running Sockets listener task detects inbound TCP/IP connection requests on all ports defined to CICS and invokes the CICS service associated with the port.
3. When the TCPIPSERVICE definition for a port has the protocol HTTP, the default transaction ID for the Web attach task is CWXN. When the protocol is USER, the default is CWXU. The transaction always executes program DFHWBXN.
4. The Web attach task matches the URL to a URIMAP definition and looks at the PROGRAM attribute to run the application program for processing the request. It runs under the default alias transaction, CWBA.

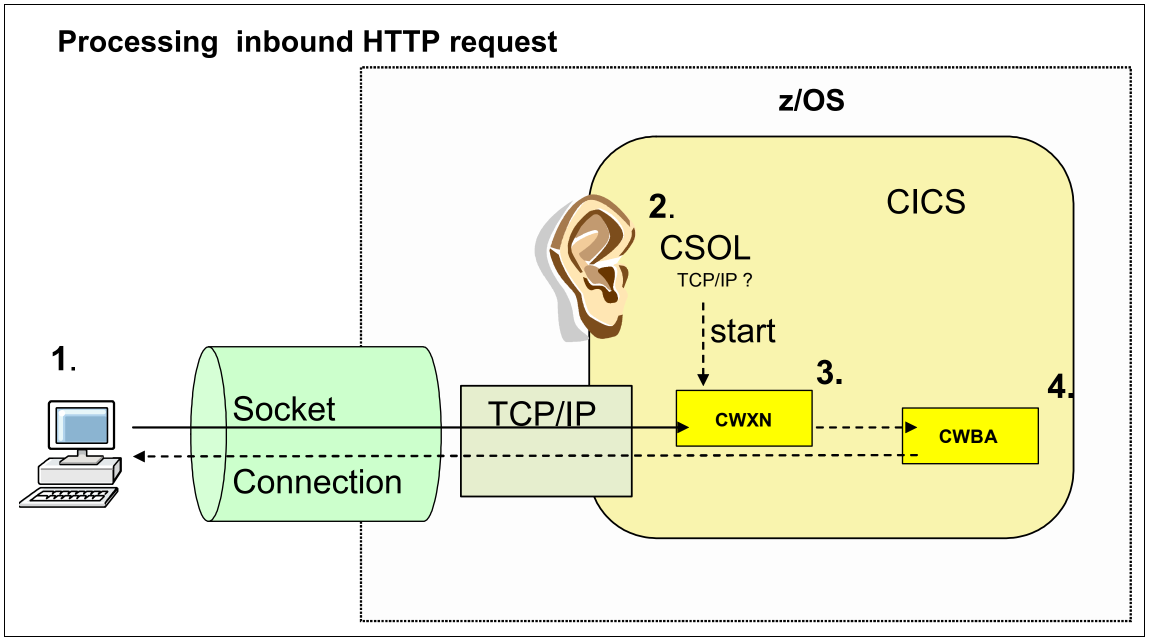


Figure 3-17 Processing inbound HTTP request

Note: The CICS TCP/IP listener is completely separate from, and not to be confused with, the TCP/IP Socket Interface for CICS, which provides an application level TCP/IP socket interface to CICS applications. It is described further in 3.6, “CICS sockets” on page 75.

3.4.2 CICS as an HTTP client

When CICS is an HTTP client, a user application program in CICS can initiate a request to an HTTP server and receive a response from it. The actions of CICS as an HTTP client are controlled by user-written application programs. An application program that makes an HTTP request and receives a response must use the EXEC CICS WEB API commands.

Processing for CICS as an HTTP client

Processing for CICS as an HTTP client takes place as a sequence of EXEC CICS WEB APIs to send the request by first establishing a socket connection, then writing HTTP Header and Body information, before transmitting the request. Response processing is also done in a sequence of EXEC CICS WEB API’s to receive the data.

Connection pooling for outbound HTTP connections

By default CICS closes client HTTP connections after an application has finished using the connection. In CICS TS V4.2, you can set up connection pooling to reuse the connection to the same host and port. CICS places the connection in a pool in a dormant state. Connection pooling is implemented by specifying a non-zero value to the SOCKETCLOSE time-out option on a URIMAP resource definition.

Note: CICS also supports non-HTTP messages. Support for non-HTTP requests is primarily intended to provide support for requests from user-written clients that use non-standard request formats. The processing that takes place for requests and the response that is provided are defined by the user. No specific support is provided for any formally defined protocols that are used for client-server communication.

3.4.3 Components for CICS web support

These are the components for CICS web support:

- Base components
  - TCP/IP Listener: The Sockets listener task detects inbound TCP/IP connection requests and invokes CICS Web support by attaching the Web attach task.
  - HTTP Handler: The Web attach task (CWXN) receives data from the web client and deals with initial processing of the request.
  - Secure Sockets Layer (SSL) support is used to provide security for the CICS Web support implementation. Refer to Chapter 6, “Security” on page 133, for details.

- Resource components for inbound http requests
  - TCPIPSERVICE resource definitions are used to define each port that you use for CICS as an HTTP server.
  - URIMAP resource definitions match the URLs of requests from web clients, or requests to an HTTP server, and provide CICS with information about how to process the requests.
  - TRANSACTION resource definitions are used to define alias transactions for HTTP request processing and linking to the business logic.
- Resource components for outbound http requests

  URIMAP is the only resource definition required for outbound http requests from CICS. This defines the security characteristics and the use of connection pooling.
- Programming components

  CICS provides EXEC CICS WEB application programming interface for handling HTTP requests and responses. Refer to Chapter 6, “Security” on page 133, for more details.

3.4.4 ATOM feeds

Atom is both a protocol and an XML format for content providers to provide XML-based web feeds of updated content. An Atom feed is a web feed provided using the Atom protocol and format. This provision of updated content is known as syndicating a web feed. Web users can subscribe to a feed, allowing them to see new content as soon as it is made available.

A web feed, sometimes just called a *feed*, is a series of related items that a content provider publishes on the internet. An Atom feed is a web feed that uses the Atom Syndication Format and the Atom Publishing Protocol. Atom comprises an XML-based format that describes an Atom feed and the items of information in it, and a protocol for publishing and editing Atom feeds.

This format and protocol are described in two Internet Society and Internet Engineering Task Force (IETF) Request for Comments documents (known as RFCs):

- RFC 4287, The Atom Syndication Format, available here:

  `http://www.ietf.org/rfc/rfc4287.txt`
- RFC 5023, The Atom Publishing Protocol, available here:

  `http://www.ietf.org/rfc/rfc5023.txt`

Content providers often deliver web feeds in an earlier format called RSS. CICS supports Atom, but does not support RSS.

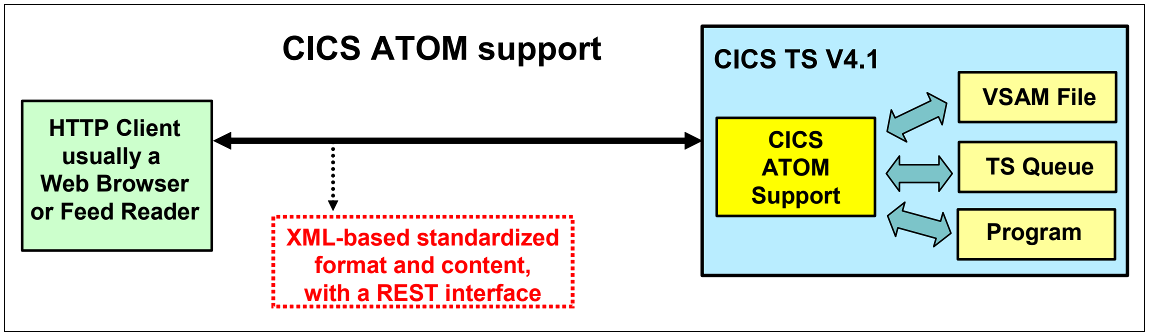
The items of information that make up an Atom feed are known as Atom entries. A content provider publishes, or *syndicates*, an Atom feed by making it available through a URL on the internet and updating it with new items. Web pages can display the items in the Atom feed, and web users can obtain the items from the feed using a feed reader or web browser.

An Atom feed might be used as part of a mashup, which is a web application that merges content from a number of data sources so that users can experience and understand the data in a new way. In a mashup, the data from the Atom feed can be handled by a widget, which is a script application that runs in a web page.

3.4.5 ATOM feeds in CICS

CICS can serve Atom feeds to web clients. The Atom feeds consist of data that is supplied by CICS resources or application programs. When you expose a CICS resource or application program as an Atom feed or collection, users can read and update the data by making HTTP requests from external client applications, such as feed readers or web mashup applications.

CICS's Atom support allows you to quickly expose a VSAM file, a TS queue, or an application program as an Atom feed (Figure 3-18).



CICS ATOM support
HTTP Client usually a Web Browser or Feed Reader
XML-based standardized format and content, with a REST interface
CICS TS V4.1
CICS ATOM Support
VSAM File
TS Queue
Program

Figure 3-18 CICS ATOM support

3.4.6 CICS ATOM support

The CICS ATOMSERVICE resource definition defines an ATOM service, feed, collection, or category document. It identifies the Atom configuration file, CICS resource or application program, and Atom binding file that are used to supply the data for the feed. URIMAP resource definitions handle the incoming requests and point to the appropriate ATOMSERVICE resource definition:

- The Atom configuration file contains XML that specifies the metadata and field names for the Atom document that is returned for this resource definition.
- The XML binding file specifies the data structures used by the resource named in RESOURCENAME, which supplies the data for the Atom document that is returned for this resource definition.

Figure 3-19 shows how CICS implements ATOM feeds.

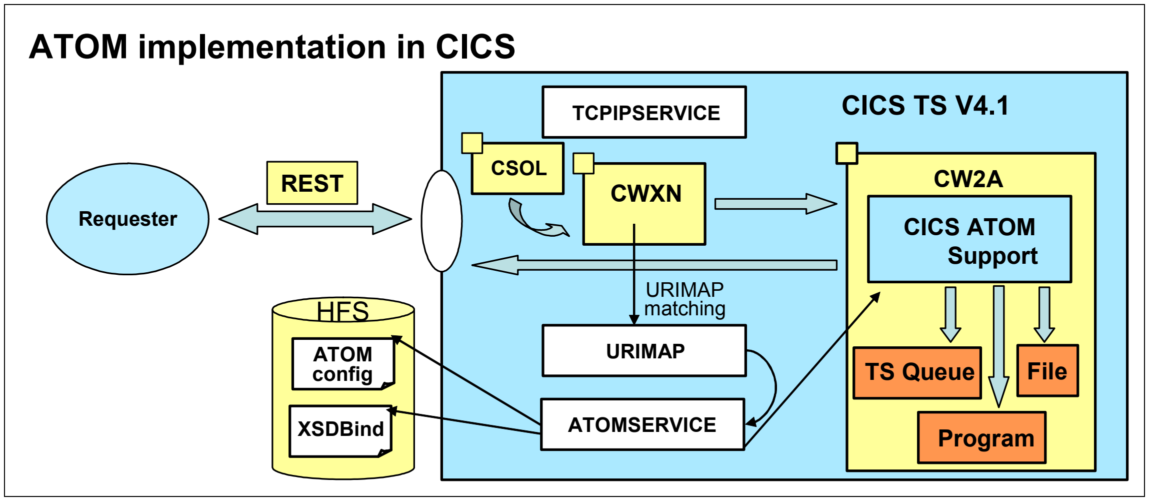


Figure 3-19 CICS ATOM implementation

3.5 WebSphere MQ

WebSphere MQ is a family of products available on all major operating system platforms. It provides an open, scalable, industrial-strength messaging and information infrastructure, enabling enterprises to integrate business processes.

WebSphere MQ provides Java Message Service (JMS) APIs and native WebSphere MQ APIs for use by clients on a wide variety of platforms. This makes it suitable for communication between a range of types of application and CICS applications.

Figure 3-20 shows a typical scenario for access of a CICS application using WebSphere MQ.

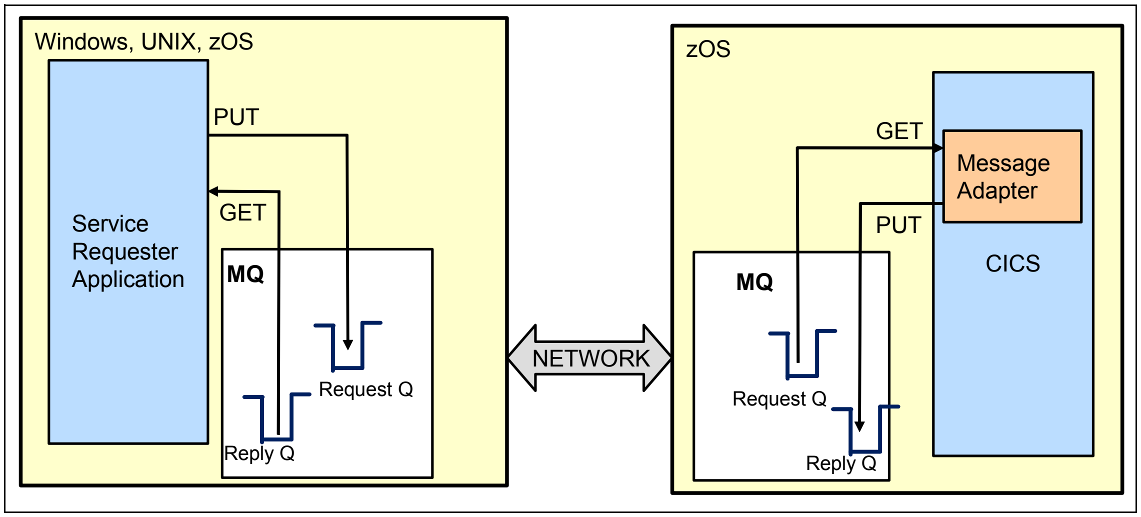


Figure 3-20 Using WebSphere MQ as a CICS integration component

Figure 3-20 shows a typical request-reply integration scenario. It uses WebSphere MQ as the transport to achieve a pseudo synchronous call. The service requester application puts a request message on a request queue and waits for the response on the reply queue. WebSphere MQ is configured to send the request message to the request queue, where a CICS application is listening. The CICS application uses the CICS-WebSphere MQ adapter to connect to a queue manager to retrieve the request from the queue. The program processes the request, formats the response message, and places it the reply queue. WebSphere MQ is configured to pass the request and reply messages between the servers, to supply the relationship information that ties the request to the reply, and to provide the assured delivery of both messages.

CICS supplies the CICS-WebSphere MQ adapter to integrate CICS applications with WebSphere MQ.

3.5.1 CICS-WebSphere MQ adapter

The CICS-WebSphere MQ adapter runs in the same address space as CICS, providing the following functions:

- It manages the connections between CICS and WebSphere MQ.
- It supports the use of MQ API calls from CICS applications.

- It provides the CICS: WMQ Bridge that allows CICS programs to be initiated by messages without having to alter the application.
- It supports triggering with the CICS trigger monitor transaction (CKTI).

3.5.2 CICS integration with MQ

This topology shows CICS integration with WMQ, using WMQ triggering:

- Topology 1: MQ API enabled CICS applications
  - Using MQ Triggering

    Figure 3-21 shows how the arrival of a message on the queue triggers a CICS application to process the message from an MQ API aware application.The flow is based on information provided in WebSphere MQ resource definitions and a CICS-provided transaction called CKTI.

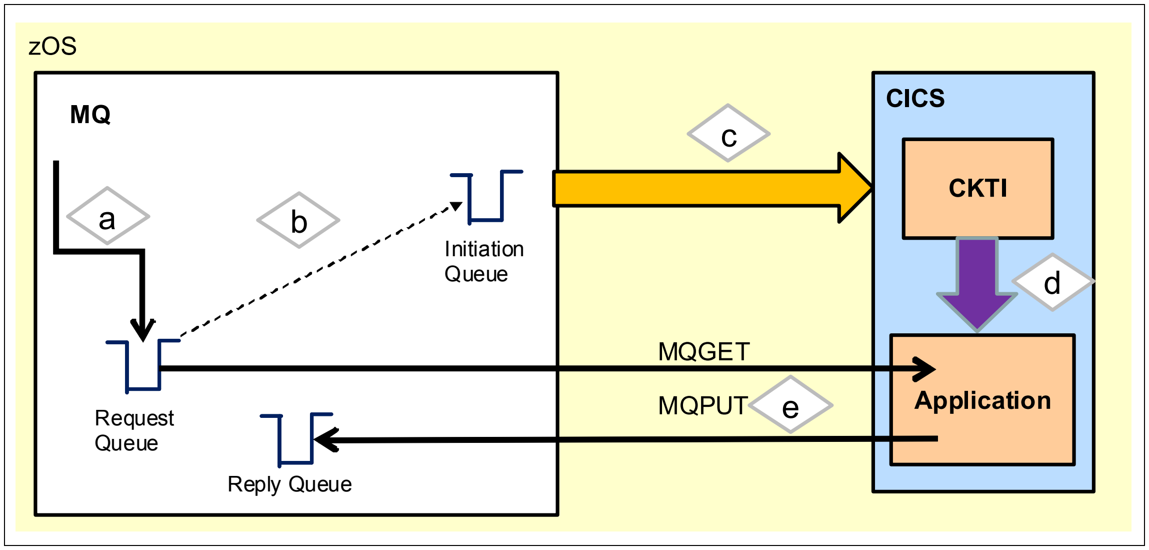


Figure 3-21 CICS MQ triggering

This is a description of what is happening in Figure 3-21 on page 70:

i. A message arrives on a request queue to be processed. This request queue has been defined to MQ with the triggering capability turned on. The definition also names the special initiation queue that will be used and the MQ process that provides the CICS transaction to be executed. The MQ process defines the CICS transaction that should be started when the trigger conditions are met.

ii. When the triggering conditions are met, MQ puts a special message (called a trigger message) on the initiation queue.

iii. CKTI monitors the initiation queues, waiting for trigger messages. The CKTI program is a long-running daemon process. It gets the trigger message from the initiation queue, and uses information from that message to begin processing.

iv. CKTI starts the WMQ application transaction defined in the MQ process associated with the trigger message. The trigger message is passed to the application program associated with the transactions.

v. The CICS application uses the information from the trigger message to open the request queue and begin processing the request messages in that queue. For throughput and performance reasons, this application will typically continue to get messages from the queue until it is empty.

Attention: The initiation queue and the process are set by the WMQ administrator rather than by the CICS administrator.

– MQ API based application with no triggering

This is a topology where the CICS application does not rely on MQ triggering. It can be a long-running transaction that remains active in the CICS region continuously, or it can be initiated in other ways. This can also be an application that is using MQ to pass messages forward without being initiated by an MQ message.

► Topology 2: CICS-WebSphere MQ bridge

CICS and MQ provide a bridge that allows non-MQ API enabled programs to be initiated by MQ messages and reply via an MQ message without making changes to the application logic. This allows applications from anywhere with the MQ network to request a CICS-based service, with no changes to the CICS application code.

The request messages have to be structured to include the standard WebSphere MQ header, the MQMD, the CICS directive, and the message body in a format that the CICS application can parse and use. The CICS directive can be as simple as a CICS program name, if the service being requested is a simple call that can be satisfied by a LINK to a CICS program passing the message body as a COMMAREA. If the request is more complex, then the MQ CICS Information Header (CIH) might be necessary to supply control options for the applications.

The WebSphere MQ bridge supports two types of bridging. The most common is the COMMAREA interface, also known as the DPL Bridge. It also supports the 3270 bridge, also known as the Terminal interface:

– COMMAREA interface (DPL bridge)

  CICS programs that are called using **`EXEC CICS LINK`**, known as distributed program link (DPL) programs. This bridge does not support channels. It can be used to run a single CICS program, or a set of CICS programs that form a unit of work.

  Figure 3-22 on page 64 shows how the CICS-WebSphere MQ bridge works.

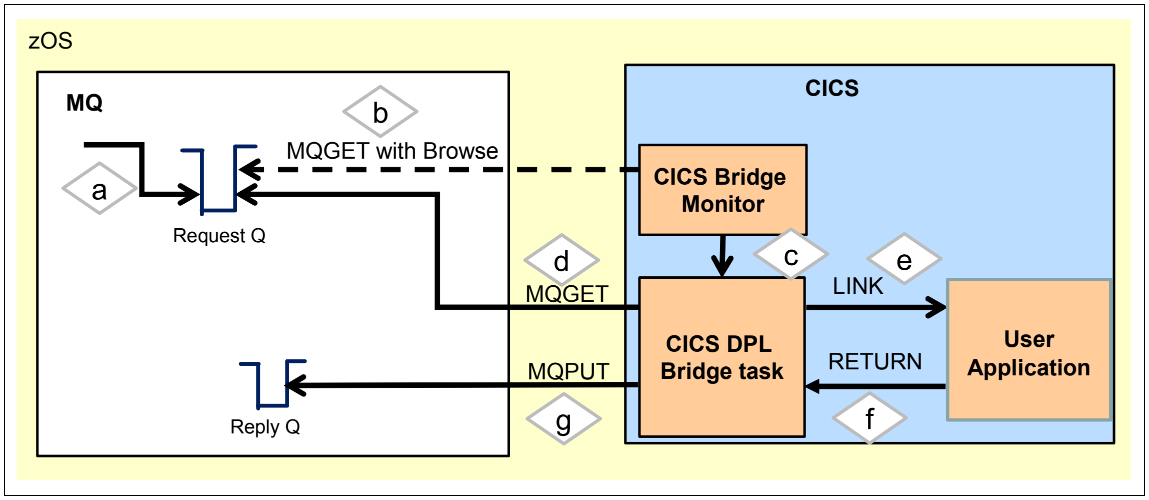


Figure 3-22 CICS-WebSphere MQ bridge workflow

This is a description of what is happenning in Figure 3-22:

i. Step a: A message arrives on a request queue to be processed.

ii. Step b: The CICS bridge monitor task, which is constantly browsing the queue, recognizes that a `start unit of work` message is waiting.

iii. Step c: The CICS bridge monitor starts a CICS DPL bridge task.

iv. Step d: The CICS DPL bridge task makes an MQGET call to remove the message from the request queue.

v. Step e: The CICS DPL bridge task builds a COMMAREA from the data in the message and issues `EXEC CICS LINK` for the program requested in the message.

vi. Step f: The program returns the response in the COMMAREA used by the request.

vii. Step g: The CICS DPL bridge task reads the COMMAREA, creates a message, and puts it on the reply-to queue specified in the request message.

After step g, if there are more messages for the same unit of work on the request queue, then the DPL Bridge task repeats steps d through g.

– Terminal interface (3270 bridge)

  3270 transactions are CICS transactions that were designed to be run from a 3270 terminal. The transactions can use Basic Mapping Support (BMS) or terminal control commands. They can be conversational or part of a pseudo conversation. You can use the CICS-WebSphere MQ 3270 bridge using the CICS Link3270 mechanism to access the CICS transaction. The request message provides the data needed to run a CICS 3270 transaction. The CICS transaction runs as if it has a real 3270 terminal, but instead uses one or more WebSphere MQ messages to communicate between the CICS transaction and the WebSphere MQ application. The workflow is similar to that shown in Figure 3-22 on page 72.

► Topology 3: Asynchronous consume

  Asynchronous consume in WebSphere MQ is a way of invoking a program by the arrival of a message instead of using the tradition MQ Triggering described above or doing a MQGET waiting for a message to arrive. It is a message-driven function directly invoked by the queue manager. The advantages of asychronous consume are that application threads are not tied up and that the application does not necessarily have to be changed to support the MQ API. The disadvantage is that it can be more CPU costly than an MQ API program.

The CICS MQ adapter supports the MQAPI functions to support asynchronous consumption. These are the MQCB (register message consumer) function, which is used to register a CICS load module as a callback routine, and the MQCTL function, which starts the message consumer. Figure 3-23 shows an example of asynchronous consumption of a message.

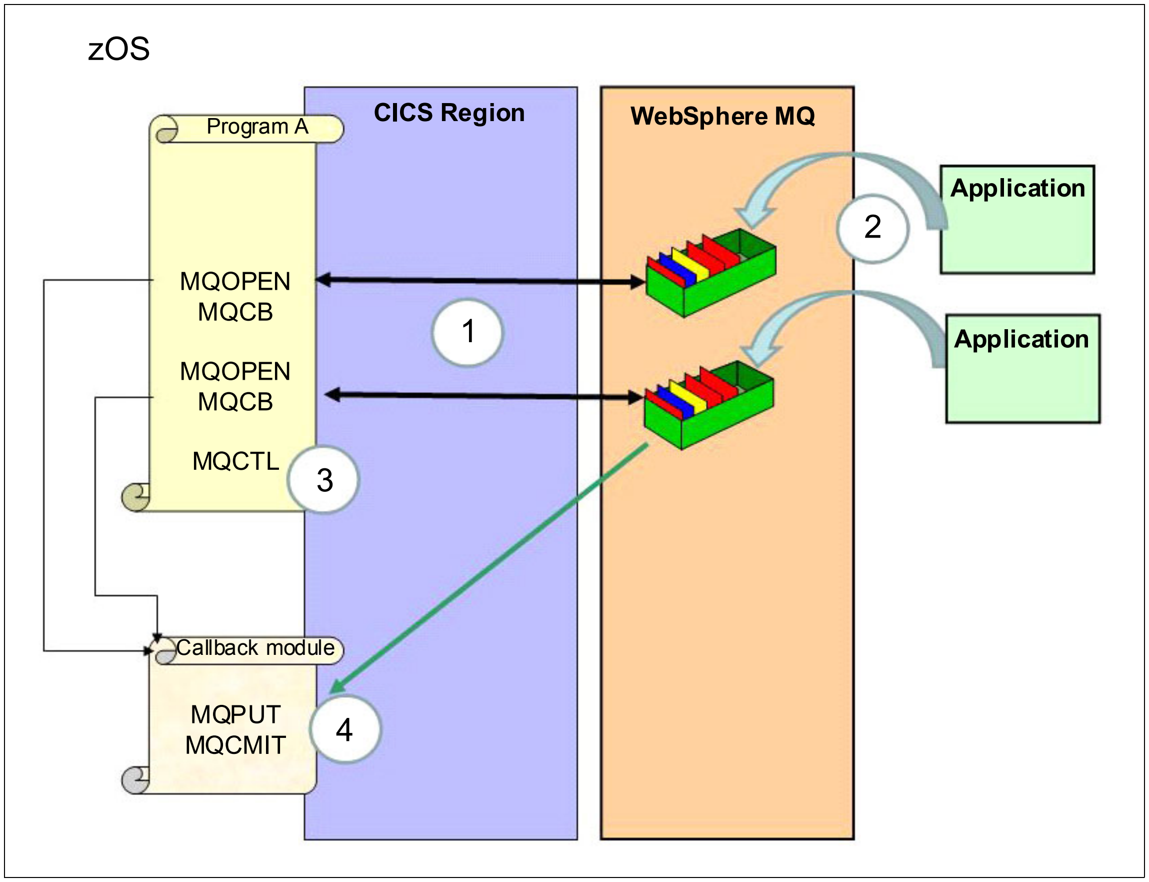


Figure 3-23 Asynchronous consume

This is a description of what is happening in Figure 3-23:

1. Program A in CICS opens queues with the MQOPEN API and registers a message consumer program B using the MQCB API.
2. The remote application puts messages on queues that need to be processed in CICS.

3. Program A uses the MQCTL API to start the message consumer.
4. The callback module starts to process the messages. WMQ passes the messages from the queues in a channel. The callback module processes the message and can optionally send the response back to a reply queue.

3.6 CICS sockets

The TCP/IP Socket Interface for CICS (also known as CICS sockets) is a feature of z/OS Communications Server that brings the TCP/IP sockets API to your CICS applications (Figure 3-24).

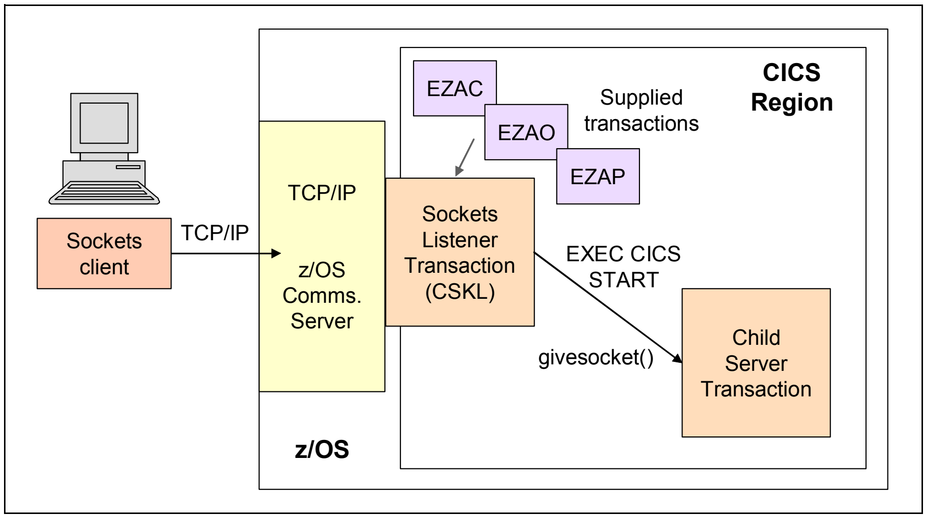


Figure 3-24 CICS to TCP/IP Sockets Interface

There are two fundamentally different models used to create CICS sockets applications:

- Iterative server

  This is the simplest model and provides in-line processing of the socket and the calls to the associated business logic. Because there is only one transaction serving the socket, all the messages sent over the socket are processed serially in the same CICS task.

- Concurrent server

  CSKL is the supplied concurrent server task and starts child server transactions for every message received. Different child server transactions can be invoked depending on the pre-defined formats of the incoming messages, and multiple instances of child server tasks can run in parallel to process multiple sockets.

The main function of the CICS to TCP/IP Sockets Interface is provided by the Sockets Listener transaction (CSKL). This is a long-running task that listens for incoming TCP connection requests on a specified port. The provided EZAC transaction can be used to configure the Sockets Listener, and in addition the EZAO and EZAP transactions can be used for operational requests. It uses a CICS Task Related User Exit (TRUE) to enable the use of native socket functions.

After CSKL receives a TCP connection request, it performs an **`EXEC CICS START`** command for a child server transaction, and passes control of the socket conversation using the `givesocket()` call. The child server transaction is the user-written socket application, and must retrieve the socket data using an **`EXEC CICS RETRIEVE`** and take control using a `takesocket()` call. This user application can be written in any language supported by CICS, including C, COBOL, Assembler, or PL/I. The design of the client/server communication is entirely up to the user application, which is responsible for all design issues including authentication and data conversion issues.

Note: The child server transaction is also capable of initiating outbound TCP connections and receiving incoming requests.

For further details, refer to the Communications Server manual, *IP CICS Sockets Guide,* SC31-8518, and to *CICS/ESA and TCP/IP for MVS™ Sockets Interface,* GG24-4026.

4

Reusing CICS applications with a 3270 presentation layer

From the late 1960s to early 1990s, the terminal devices such as the 3270 were the most popular type of client for CICS applications. The presentation and business layers in these CICS applications were optimized and limited to the capabilities of the terminal, often requiring more than one screen interaction to validate and accumulate the necessary information before finally processing the request against a database.

This resulted in some CICS applications combining the presentation and business layers in the same program. This presents challenges when needing to reuse the business layer to support new client types. In this chapter, we examine three technologies that enable the reuse of these types of applications without requiring them to be significantly changed.

4.1 Terminal-orientated CICS applications

CICS applications that support clients that are 3270 devices are sometimes referred to simply as 3270 programs. They were designed to be used by an IBM 3270 Display Station or similar buffered terminal device. Invocation usually corresponds to a single interaction in an end-user dialog, starting with receipt of a message from the terminal and ending with transmission of a reply message to the same device. Input data from the terminal device is carried in a datastream, which the application acquires through a **`RECEIVE`** command. After processing, an output datastream is transmitted back to the terminal device through a **`SEND`** command. Terminal-oriented programs must be capable of analyzing device-specific input data streams and building output data streams to be transmitted to the terminal.

CICS TS also provides a service known as Basic Mapping Support (BMS), which simplifies application programming for terminals such as the IBM 3270 Display Station. This enables the programmer to define a static layout for each screen to be displayed, with identified fields for dynamic content acquired through a **`RECEIVE MAP`** command. This in turn causes BMS to analyze the datastream and to return record-formatted data to the application. Similarly, the application presents output data in record format using a **`SEND MAP`** command, which causes BMS to build an output datastream for the terminal. BMS is widely used because it frees the application programmer from needing knowledge of device specifics and enables applications to be device-independent to some degree.

A pseudo-conversational model is normally associated with terminal-oriented transactions. A pseudo-conversational sequence of transactions contains a series of transactions that look to the user like a single conversational transaction involving several screens of input. However, each transaction in the sequence is in fact a single transaction that handles one input, sends back the response, and then terminates.

Access to terminal-oriented programs

Many programs remain that do not have such a clear separation of concerns as COMMAREA programs, combining presentation layers and business into a single program, such that the business layer cannot be reused easily by other client types.

In this section, we focus on three technologies that enable a 3270 application to be re-used:

- CICS Front End Programming Interface (FEPI)
- IBM Rational Host Access Transformation Services (HATS)
- CICS Link3270 Bridge

Figure 4-1 shows the differences between these technologies at a high level.

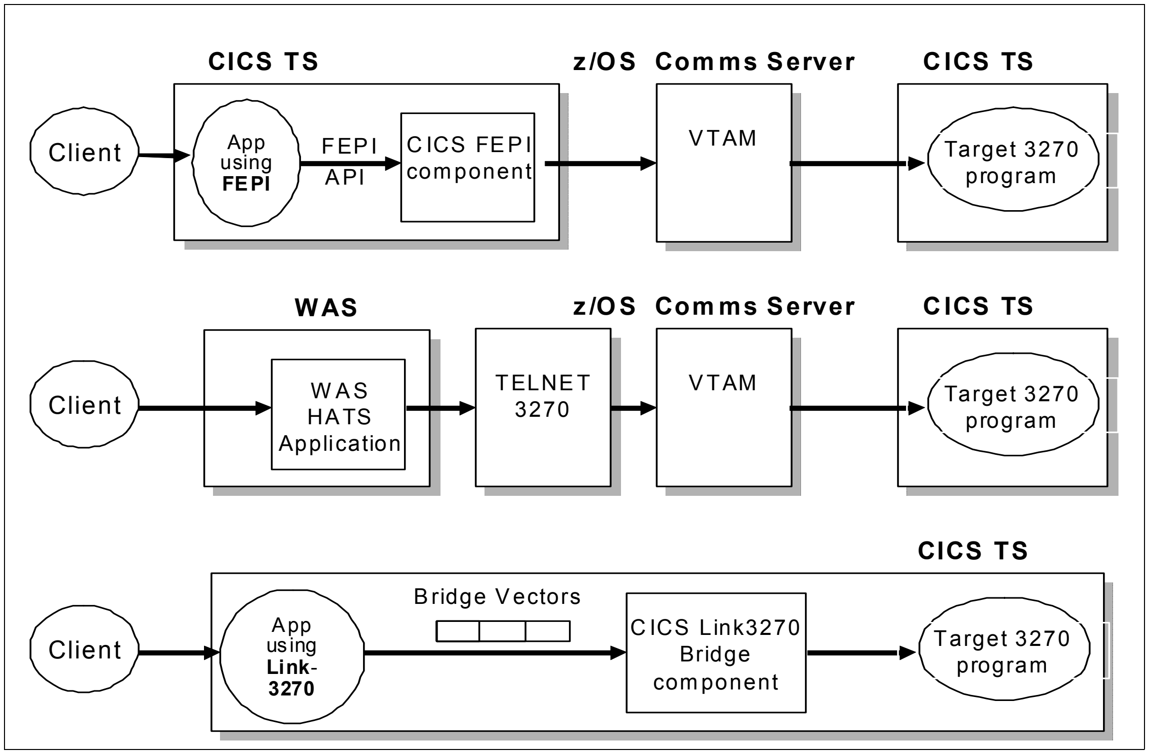


Figure 4-1 Comparison of FEPI, HATS, and the Link3270 Bridge architectures

In Figure 4-1, the client at the top of the diagram connects into CICS TS using an integration technology such as web services or CICS TG. A CICS application is invoked, which uses the FEPI API to tell CICS to communicate with the target 3270 program through VTAM® (Virtual Terminal Access Method) by emulating a 3270 terminal.

The client in the middle of Figure 4-1 connects over HTTP to an application running in WebSphere Application Server (WAS) on either a distributed platform or z/OS. The application, which is generated via HATS tooling, emulates a 3270 terminal and connects through VTAM to the target 3270 program.

The client at the bottom of Figure 4-1 on page 79 connects into CICS TS using a technology such as web services or CICS TG. A CICS application is invoked, which in turn invokes the CICS Link3270 bridge, passing data in the form of bridge vectors. The CICS Link3270 bridge bypasses VTAM and interacts directly with the target 3270 program.

FEPI

FEPI is an API provided by CICS TS. It allows a CICS application to behave as if it were a user interacting with a CICS 3270 application via a terminal. The CICS FEPI component acts as a virtual terminal. It communicates via VTAM to the target 3270 program, in the same way as a real terminal would. The CICS application using the FEPI API sends commands such as “Position the cursor at point X on the screen then enter the text ABC”. The benefit of this approach is that the target 3270 program is being called through VTAM as it would be from a real terminal. Therefore, the behavior of the target 3270 program will be as it is for any other terminal.

HATS

HATS provides the ability to create a new presentation layer, such as a web front end, to an existing CICS 3270 program. Tooling is provided to enable the user to easily generate modern presentation layers that map to the input expected by the target 3270 program. The tooling generates a runtime component, which runs in an environment such as WebSphere Application Server (WAS). A client can connect to WAS from a web browser. The generated WAS application would parse the clients input, then interact with the target 3270 program through a telnet 3270 server, which in turn communicates with VTAM. From the perspective of the target 3270 program in CICS, this is just another terminal connecting via VTAM.

CICS Link3270 bridge

The CICS Link3270 bridge allows an application in CICS TS to call a CICS 3270 application, and is similar in its aims to FEPI. The difference in the approaches is that the Link3270 bridge removes the need for requests to flow out of CICS TS via VTAM to the target 3270 program. Instead, the Link3270 bridge component interacts directly with the target 3270 program, so for example, when the target 3270 program issues a RECEIVE call to obtain data from the terminal, this call is handled by the Link3270 bridge component, which passes the required data to the target 3270 program. The invoking application passes data to the Link3270 bridge in a format known as *bridge vectors*. The bridge vectors are used by the Link3270 bridge to build the information requested by the target 3270 program. The benefit of this approach is the removal of the need to interact via VTAM. The potential drawback is that the target 3270 program is now being driven in a different manor from its original design, and there are some CICS API restrictions to which the target 3270 application must adhere. See the “Link3270 programming considerations” section of the *CICS TS External Interfaces Guide,* SC34-6449, for further details.

4.2 Technology options

In this section, we explore each of the three technologies in further detail.

4.2.1 CICS Front End Programming Interface

The Front End Programming Interface (FEPI) is a CICS API. The function is called a front-end programming interface because it enables you to write CICS application programs that drive other CICS 3270 programs. That is, it provides a front end to those programs. The interface simulates the terminals that the other programs use. The ability to drive these 3270 programs from another CICS program allows the existing 3270 programs to be used in different ways without changing them. An application could make calls to multiple 3270 programs to provide a new service. The existing 3270 programs can be on the same system as the simulating program or on a different system. If the newly written applications do not contain presentation logic themselves, then they could be exposed as services using one of the architectural styles and technology choices discussed throughout this book.

For further information about FEPI, refer to the CICS TS Information Center, in particular, the *Front End Programming Interface User’s Guide,* SC34-7169.

4.2.2 IBM Rational Host Access Transformation Services (HATS)

If your goal is to provide a more modern, web-based interface to your existing 3270 programs, HATS can help achieve this. HATS assists with the creation of a new presentation layer for the existing 3270 programs and does so without requiring those programs to be changed, which minimizes the risk involved.

The HATS solution provides a quick and easy way to replace 3270 displays with a simple point-and-click interface and provides the tools needed to quickly and easily transform 3270 programs to web, portlet, rich client, or mobile device user interfaces, or to create a web service front end to a 3270 program.

HATS has a development component called the HATS toolkit. The HATS toolkit features a wizard-based development process for creating HATS applications to transform existing 3270 programs. The development process is similar regardless of whether you are extending your CICS 3270 programs to the web, portal, a mobile browser, a rich client, or as web services. The HATS solution enables you to tailor your application to a specific set of end users, hide unnecessary information, organize data into tables, or display only required input fields. You can also provide drop-down lists of valid values for an input field, change the size and location of text, and provide navigation buttons to reduce data entry errors and increase productivity.

There is no specialized HATS runtime server. For a web interface, all of the necessary runtime information is deployed into an Enterprise Archive (EAR) file and runs in WebSphere Application Server or WebSphere Portal. For a rich client interface, the necessary runtime information can be generated to run in an environment such as the Eclipse Rich Client Platform. The generated code takes care of the interaction with the target 3270 program.

For further information about HATS, refer to this website:

http://www.ibm.com/software/awdtools/hats/

4.2.3 CICS Link3270 bridge

The Link3270 bridge provides a callable interface to allow you to run 3270-based CICS transactions without emulating a 3270 terminal. This has the benefit that 3270-based user transactions can be retained within CICS TS where necessary, and access to them exposed via a more simple callable interface.

The 3270 terminal and end user are replaced by an application program, known as the bridge client application. Commands for the 3270 terminal in the 3270 user transaction are intercepted by CICS TS and replaced by a messaging mechanism that provides a bridge between the client application and the 3270 user transaction.

This mechanism provides a link interface that can be accessed using a distributed program link (DPL), an External Call Interface (ECI) request, or an External CICS TS Interface (EXCI) request. The bridge client requests services of the Link3270 bridge using COMMAREA messages in a prescribed format called bridge vectors, and the Link3270 bridge returns results to the bridge client in formatted messages.

The Link3270 bridge provides an interface to enable 3270-based CICS transactions to be invoked using a LINK request, and without the facilities of a 3270 terminal. As such, it is not a means of providing web access (because there is no direct means of generating HTML), but it is an important enabling technology because it allows a client module to link to existing 3270 transactions. The client module could itself be exposed to external clients that need access to the business logic.

The client application uses the Link3270 bridge to run 3270 transactions by passing a COMMAREA that identifies the transaction to be run and contains the data used by the user application. The response contains the 3270 screen data reply.

For further information about the Link3270 bridge, refer to the CICS TS Information Center, in particular the *External Interfaces Guide*, SC34-7168.

4.3 Tooling

While HATS comes with its own tooling, the Link3270 bridge and FEPI require a programmer to write a program that builds the interactions between the invoking application and the target 3270 application. Performing this process manually can be complex and time consuming, so it is worth considering tooling that can assist with this process. IBM provides Rational Developer for System z (RDz), which contains a component called the Service Flow Modeler, which can be used to simplify the development of applications that need to interact with 3270 programs.

As an example, perhaps you have two 3270-based terminal applications that you want to call serially to compose a new CICS application, and you want that CICS application to be made available as a web service. The Service Flow Modeler provides tooling to record the terminal interactions with the target 3270 programs, model the new application, and generate runtime code and artifacts that can be deployed into CICS TS. The result is a web service enabled CICS application that uses the Link3270 bridge or FEPI to invoke the 3270 applications that it relies on. Use of this tooling can make development of such applications considerably easier than manual coding. This tooling is discussed in the following section.

RDz Service Flow Modeler

The Service Flow Modeler (SFM) provides the tooling for generating flows that can be deployed into CICS TS. Within CICS TS is a component called the Service Flow Runtime (SFR), which provides the runtime environment for the generated flows.

It is important to note that SFM can generate flows that use FEPI, HATS, or the Link3270 bridge technology to access terminal-oriented applications that contain 3270 presentation logic.

SFM enables distributed applications to make business requests of existing CICS 3270 and COMMAREA applications as callable services. It enables customer-written applications to integrate seamlessly with business-critical 3270 and COMMAREA applications by generating service flows that contain a web service interface.

SFM provides modern graphical tooling based on the Eclipse platform. The tooling provides a means to simplify the re-use of existing 3270 and COMMAREA applications within business processes by assisting with tasks such as these:

- Interactions with terminal-based programs can be recorded in a simple manner by following a guided wizard.
- Application flow, conversion, and integration can be orchestrated within the tooling.
- The completed business process flow, including terminal interactions, can be generated as code for the Service Flow Runtime in CICS TS.
- The generated business process can be automatically deployed into CICS TS.
- Support is provided for exposing business processes as web services inside CICS TS.

Note: Running the business process inside CICS TS can be important for performance, as a client invoking the business process over the network only makes one invocation to the business process, which could then call multiple CICS programs sequentially. An alternative approach is to run the business process outside of CICS TS and make a call to CICS TS for each CICS program that needs to run, but this approach will make more calls over the network, which can be costly.

For further information about the Service Flow Modeler, refer to the “Enterprise Service Tools for Web services and SOA” section of the IBM Rational Developer for System z information center:

`http://publib.boulder.ibm.com/infocenter/ratdevz/v8r0/index.jsp`

CICS Service Flow Runtime

CICS Service Flow Runtime (SFR) is a run time that enables business processes created with SFM to be deployed and run in a CICS environment. SFR is a strategic solution that is used to avoid being forced into programming-intensive solutions that are prone to error, especially when the host CICS applications are changed for maintenance or upgrade.

SFR uses SFM-generated adapter services to provide the sequencing of 3270 screens in a CICS application. Thus, when a service request comes to CICS from a distributed application, SFR navigates the appropriate 3270 screen sequences, formulates a consolidated response, and sends a single service response to the requester. Figure 4-2 shows a conceptual view of a generated service flow deployed into CICS.

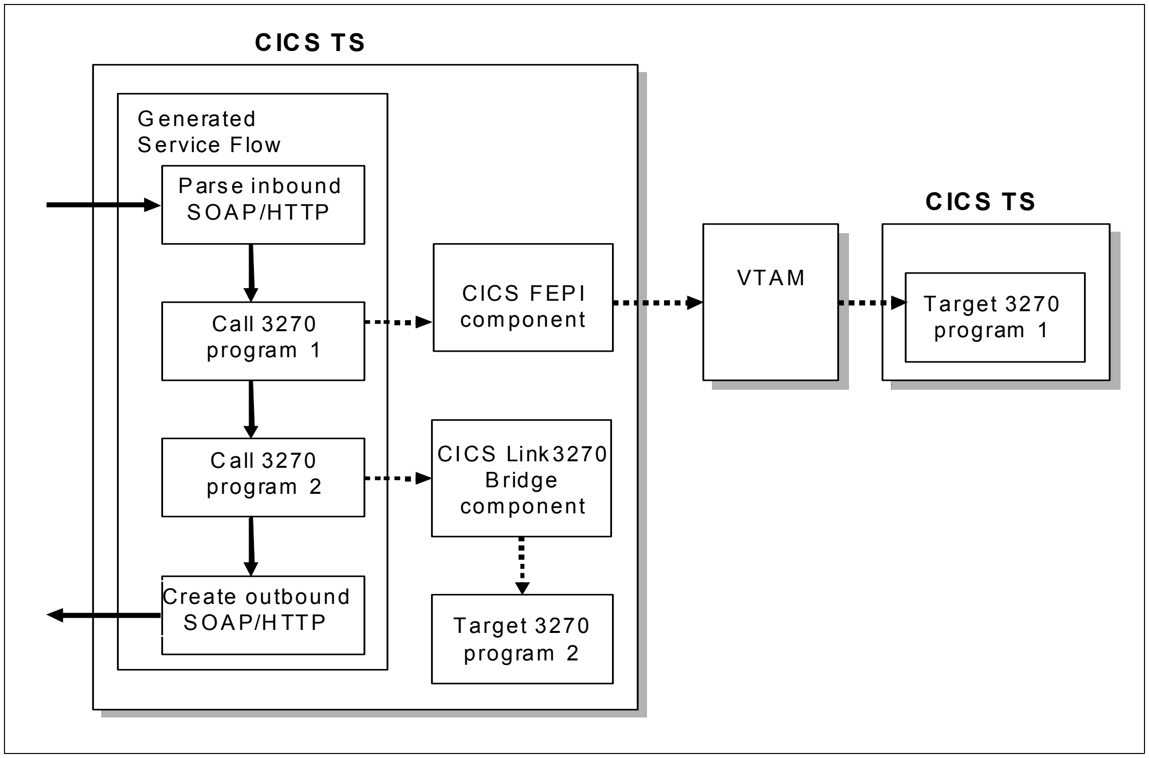


Figure 4-2 A conceptual view of a generated service flow deployed into CICS TS

In Figure 4-2, CICS TS exposes a web service that can be invoked from any web service enabled client application. On receipt of a web service request, the message is parsed by CICS TS and converted into a data structure suitable for use in a CICS program. The service flow then makes a call to a 3270 program using FEPI. On return from that call, the service flow makes a second call to a 3270 program, this time using the Link3270 bridge. Finally, a response message is built by the service flow, converted to a web service response message by CICS TS and returned to the invoking application.

For further information, refer to the CICS TS Information Center, in particular the *Service Flow Runtime User's Guide,* SC34-6913.

Part 2

Qualities of service

In Part 2 we provide an in-depth analysis of the important factors that are likely to affect your choice of integration technology. This includes factors that affect application interfaces, security, transactional scope, high availability, and scalability.

5

Application interfaces

This chapter considers the application interface issues that might impact the access technology choice. We first discuss the common interface types and message formats that are used to access CICS applications. Then for each of the strategic CICS integration technologies we review how each technology provides the client interface through a set of functions, including these:

- Transport and protocol adaptors
- Operation identification
- Message adapters
- Message exchange patterns

5.1 Application interface issues

An interface is a point of interaction between two components. These components communicate using input and output messages that are exchanged using a transport protocol.

This point of interaction is commonly defined as a combination of these properties:

- An operation, function, procedure, or method that identifies the point of interaction. We refer to it as an *operation* in this chapter.
- Input or output messages that are exchanged when accessing this operation. The messages are often stored within an envelope such as a WebSphere MQ message, an HTTP request, or a SOAP envelope.
- Message formats that define the message structure.
- Message exchange patterns that identify the type of interaction with the operation.
- A transport protocol that is used to communicate between the components.

Newer technologies such as web services formulate these interaction properties within a service definition document (a WSDL file, for example), enable automatic interface creation through tooling, and simplify component integration by using standards. Older technologies, such as the CICS sockets support, use a proprietary approach, relying on local programming skills.

Most implementations use the concept of headers to supply message metadata or quality of service information. This ranges from a user-defined private CICS sockets implementation, to a proprietary CICS ECI flow, to standard HTTP transport headers and SOAP envelopes.

The most common use of CICS is as a server or service provider. The application interface is determined by the CICS application that is already in place most of the time. The interface granularity and coupling requirements are important factors in determining the most appropriate access technology to use.

CICS can also be a client or service requester. In this role the application interface is normally determined by the server or service provider.

The application interface must address the following challenges:

- Transport handling
- Operation identification
- Message envelope extraction from the transport layer

- Message serialization and deserialization

  Serialization is the process of converting a data structure or object state into a format that can be stored or transported across a network connection link and "resurrected" later in another computer environment. Deserialization is the reverse process.

- Message exchange pattern identification

These concerns are mostly addressed by the native CICS infrastructure and associated application development tooling that is used to generate a *message adapter*.

Important: Most of today's CICS applications have been in place for a long time. Application interface modernization with no modification to the existing application interface will most likely require an adapter. When CICS is a client, an adapter or proxy is also likely to be required to enable the interaction with the server.

5.1.1 CICS program interfaces

CICS supports a rich set of interfaces within which these are the two principal ones:

- 3270 interface
- LINK interface

3270 interface

CICS applications written to interact with a user via a terminal can use the 3270 interface. The characteristics of the 3270 interface are as follows:

- The operation is imbedded within the 3270 data stream. It is typically a 3270 function key representation.
- The message is serialized as a 3270 data stream.
- The message format is dynamic and mapped to 3270 screen layouts. The CICS BMS function supplies an adapter that allows an application to use a language structure message interface rather than a 3270 data stream message.
- The type of interaction is conversational (one long CICS transaction that converses with the 3270 terminal interface) or pseudo conversational (each interaction with the 3270 terminal is a separate transaction).
- The transport is SNA.

See "Access to terminal-oriented programs" on page 78 for information about how you can re-use applications that are currently written with a 3270 interface.

Note: For the rest of this chapter we concentrate on the application interface considerations for applications that use the LINK interface.

LINK interface

The CICS LINK interface allows one program to transfer control to another in a synchronous manner and continue operation after the called program has returned. It is used for communication between CICS programs running on the same CICS server or in the form of a Distributed Program Link (DPL) when the target program is running remotely. The LINK also allows non-CICS programs running on different platforms to interact with CICS applications.

An advantage of DPL is that you can write an application without knowledge of the location of the requested programs. That is, it offers a location independence. It also facilitates communication between heterogeneous programming languages.

The characteristics of the LINK interface are as follows:

- The CICS program name represents the operation.
- The CICS message format is a COMMAREA or channel.
- The message format is normally defined by a language structure.
- The type of interaction is similar to a Remote Procedure Call (RPC).
- Many different transport mechanisms can be used.

Additional optional properties of the LINK can be used to control quality of service information, for example:

- The CICS transaction code to be used to identify the workload associated with the CICS program.

  This is an optional property that applies to a DPL client type of LINK. A best practice is to always identify a workload to CICS. This enables CICS optimizations and capabilities such as workload management, location transparency, security, auditing, or monitoring. Although it can be set on the LINK, another best practice is to delegate the setting of this property to the IT system administrator.

- The CICS commit transactional behavior (that is, whether the DPL is part of a distributed transaction).

  The best practice is to leave the transactional behavior decision to the IT system infrastructure.

- The CICS region target location identification.

  Again, it is best left to the IT system infrastructure to determine the target CICS region based on static or dynamic criteria.

LINK and DPL APIs are provided for a range of different types of access, including these:

- Command-level APIs for programming languages such as these:
  - COBOL, C, and PLI
  - Object-oriented classes for C++
  - JCICS classes for Java
- SOAP web service interactions, which map web service requests to LINK interactions.
- WebSphere MQ through the DPL Bridge function, which maps a message to a CICS LINK interaction.
- CICS TG through the ECI or CCI support.
- WOLA support, which maps a CCI request to a CICS LINK interaction.

User-coded adapters can also supply LINK or DPL support for other specific requirements. They typically use CICS APIs to simplify the coding.

COMMAREA message

The CICS COMMAREA message is a byte area with a maximum length of 32 kb. The message is common to both the request and the response, which requires its length to be the maximum between them. This is usually the response message length. Determining and handling the real length of significant data within the COMMAREA can prove to be difficult, so in most cases the length is set to the maximum length.

The message format is usually defined by a language structure where fields are positional. However, CICS does not impose any restriction on the format, and therefore other formats such as XML can also be used.

The COMMAREA is a multipurpose and fixed message interface. As such, it imposes redefinitions of the same physical area, the most trivial being the request and response layouts. To solve this problem, some implementations define two separate request and response areas in the same unique COMMAREA layout. As the interface evolves, this introduces a level of complexity that might not be acceptable today.

Channel message

Over time, CICS LINK interface requirements have evolved and the COMMAREA is often considered to be too restrictive. As a result, CICS introduced a channel message, which adds flexibility to the length and organization of the data passed on a LINK request.

Containers are named blocks of data designed for passing information between programs. Programs can pass any number of containers between each other. Containers are grouped together in sets called channels. Figure 5-1 shows an example channel and an analogy with an XML document.

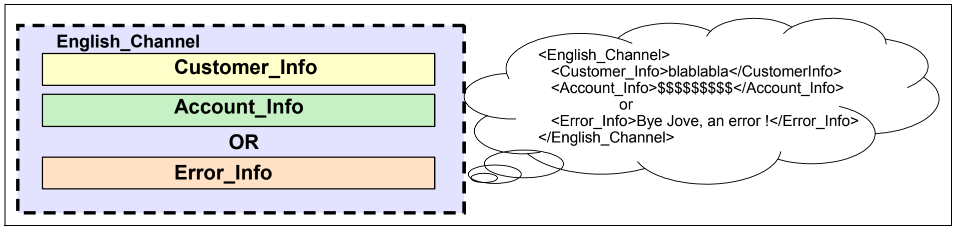

English_Channel
Customer_Info
Account_Info
OR
Error_Info
<English_Channel>
<Customer_Info>blablabla</CustomerInfo>
<Account_Info>$$$$$$$$$</Account_Info>
or
<Error_Info>Bye Jove, an error !</Error_Info>
</English_Channel>

Figure 5-1 CICS channel message representation

The channel interface offers a rich EXEC CICS CONTAINER API that can be used to implement LCRUD services:

- *L*ist or discover the current set of containers in a channel.
- *C*reate or PUT a container in a channel.
- *R*ead or GET a container from a channel.
- *U*pdate or PUT replace a container in a channel.
- *D*elete a container from a channel.

The CICS infrastructure provides a set of services that simplifies the programming effort:

- Channel life cycle and memory allocation/de-allocation
- Monitoring and statistics data on channel usage
- Codepage conversion for character containers
- Message size optimization, for example, where read-only containers are not returned to the client application

The channel message enables a more structured interface so that different types of business data can be encapsulated within an appropriately named container. It is common to have a different set of containers for the request and response messages. The benefits of using channels and containers include:

- More readable (understandable) code and interface
  - For easier maintenance
  - For better productivity
- An interoperable interface that is CICS implementation neutral
- Reuse of the container structure definitions within multiple channel interfaces

From a business perspective the channel message offers the best interface. From a runtime perspective, however, it has a higher cost of infrastructure than a

COMMAREA interface. The following section gives guidance on how to chose between the different LINK and message options.

LINK use cases

Figure 5-2 shows a typical LINK interaction.

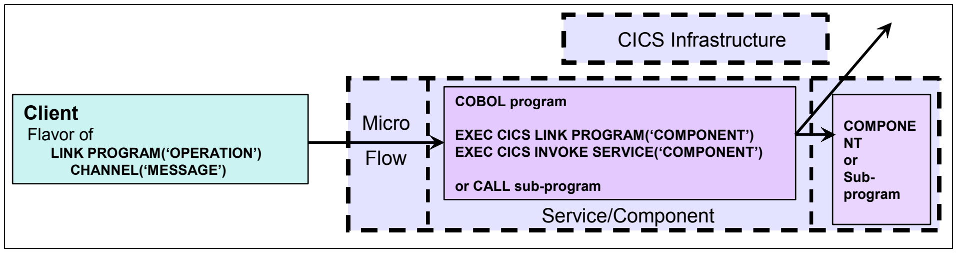


Figure 5-2 Example LINK interaction

Figure 5-2 shows the following components:

- The client that uses a flavor of a LINK request such as a CICS DPL, ECI request or web service request.
- A *micro flow* that performs a sequence of LINK requests, thus increasing the granularity of the interface exposed to the client. A micro flow can be implemented as a CICS service flow using the RDz tooling (see "RDz Service Flow Modeler" on page 83). This is a convenient way to expose coarse-grained services from multiple fine-grained LINK interfaces or even 3270 interface programs.
- The arrows on the right in Figure 5-2 illustrate access to CICS or non-CICS programs.
- CICS components and sub-programs: A component typically exposes a business interface, while a sub-program typically exposes an IT interface.

The primary use case of a channel is to expose a message interface to external clients or consumers. The business logic itself is likely to become a client or consumer of other exposed interfaces. These can be CICS or non-CICS programs. Typical CICS programs use a LINK or low-level language facilities such as a COBOL call to access CICS interfaces. Business logic should not contain access technology related code, such as WebSphere MQ, HTTP or web services. The best practice is to use an intermediate adapter dedicated to the remote interface access technology. Figure 5-2 shows an INVOKE SERVICE call, which was introduced with CICS TS V4.

An EXEC CICS LINK couples the service call to a CICS program. The INVOKE SERVICE call currently offers the choice between a CICS LINK or a web service requester implementation. The type of access is fully transparent to the application code. The decision to use one or the other is delegated to a CICS deployment action. This reduces the coupling between the business logic and the technology used to interact with other service components.

For CICS-to-CICS interface access, the scope of solutions is a LINK channel or a LINK COMMAREA or a local low-level language call to function. They range from high quality to average quality, and average CPU cost to minimum CPU cost.

The best practice is to use a selection of different types of LINK interactions based on the granularity of the accessed interface:

- A coarse-grained interface or business service interface is the place for business agility optimization, so normally the best choice is the LINK channel interface. It can be superseded by an INVOKE SERVICE.
- A fine-grained interface is the place for a low-level language call. A fine grained interface is likely to be a technical interface rather than a business function. This is the place for IT optimization.
- A medium-grained interface could be the place to use a LINK with COMMAREA, which balances application development flexibility with performance factors like CPU cost and memory optimization.

5.1.2 EBCDIC message conversion

CICS TS runs on an EBCDIC platform and so the LINK interface expects EBCDIC messages. When sending data from distributed systems that use different character sets, it is common practice to perform the conversion within CICS.

CICS implements different data conversion techniques dependent on the type of client:

- CICS uses native z/OS services to handle conversion of UNICODE messages for web service requests.
- CICS supplies the DFHCCNV conversion service for CICS TG COMMAREA messages. A PROGRAM entry in the CICS DFHCNV table supplies the conversion information. This is a system programming task that requires synchronization between the development and the production support teams. Unicode conversion is not supported.

  Note: You can create generic templates that apply to multiple programs by specifying a prefix that can be matched against multiple program names.

- CICS web support provides data conversion support with its WEB SEND/RECEIVE APIs.
- The container resource manager supports data conversion services for character data containers with its GET and PUT CONTAINER APIs.

Note: The best practice is to exchange character data on DPL requests.

5.1.3 Service interfaces

For SOA-based solutions, the application interface is abstracted into a service interface. The first and immediate benefit of this interface is interoperability, where any client implementation can easily access the service. This is the IT view of SOA. A second benefit is service reuse, where services can be composed into business processes. This is the business view of SOA.

The SOAP web services world is composed of sub-programs or methods and is a perfect fit with the CICS LINK interface. This implies extended capabilities that enable different qualities of service for the interaction, but also some complexity.

Web 2.0 technologies have emerged over recent years that extend SOA to web-based applications using technologies such as REST and POX:

- REST is a style of a web architecture that describes simple interfaces that transmit domain-specific data (whose patterns are identified using conventions) over HTTP (whose methods are used as action verbs) without any additional messaging layer such as SOAP.
- Plain Old XML (POX) is a term used to describe basic XML. It is coined from the phrase *plain old Java object* (POJO) and is in contrast to a more layered XML usage, such as that used in SOAP-based interactions.

Important: Web-oriented architecture (WOA) is a style of software architecture that extends service-oriented architecture (SOA) to web-based applications and is sometimes considered to be a light-weight version of SOA.

CICS supports both SOA-based and WOA-based solutions. These are the characteristics of a “modern” CICS service interface:

- The operation is abstracted (typically using SOAP or REST artifacts).
- The message is serialized as an XML document or JSON flow.
- The message format is abstracted using XML or JSON.
- The transport is HTTP or WebSphere MQ.

5.2 CICS inbound access architecture

Figure 5-3 illustrates the CICS architecture that supports inbound access to CICS.

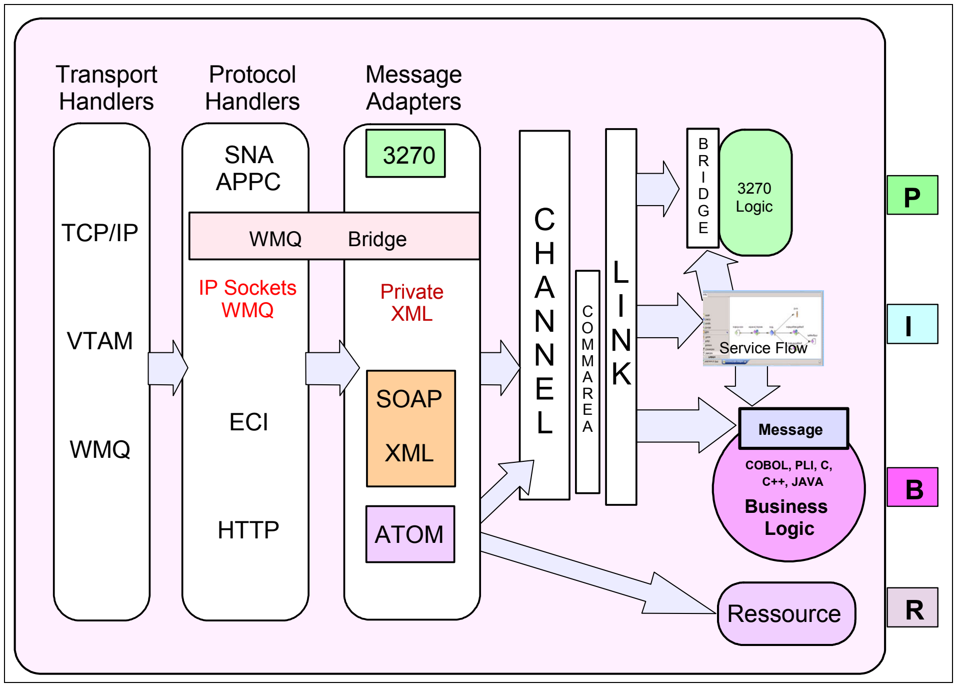


Figure 5-3 CICS inbound access architecture

On the right of Figure 5-3 you can see the different application logic layers:

- *P*resentation
- *I*ntegration
- *B*usiness
- *R*esource

The CICS Service Flow run time runs service micro-flows generated from IBM Rational Developer for System z (RDz). These flows then use the LINK or LINK3270 bridge interfaces.

The CICS infrastructure services are grouped across three layers:

- Transport handlers

  This is where the low-level transport is handled. From an application perspective, TCP/IP, VTAM, and WebSphere MQ interfaces are handled transparently by CICS.

- Protocol handlers

  This is where the operation and the message envelope are processed. The operation is identified and translated into a CICS implementation, that is, transaction code and program name. The message envelope is extracted from the transport protocol.

  CICS supplies native and transparent support for high-level protocols such as SOAP over HTTP or WebSphere MQ. TCP/IP sockets is a low-level protocol that requires more complex application code. Medium-level protocols such as HTTP benefit from simple CICS API support.

- Message adapters

  The CICS infrastructure supplies a SOAP/XML message adapter and a WebSphere MQ DPL bridge adapter.

  CICS Atom feed support adds a new capability to the standard CICS program interfaces. Rather than accessing a program, it is more likely that an Atom feed accesses a data resource. The ATOM specification RFCs define a standard XML format, along with a REST style http interaction. The CICS infrastructure uses these standards to invoke the CICS resource manager handling the type of data resource associated with the feed. Currently, CICS supports CICS Temporary Storage and VSAM data access.

  Atom feeds can also be used to access CICS programs. This requires some programming effort. The CICS Dynamic Scripting feature supplies a simpler model.

Note: WOLA is not represented in Figure 5-3. WOLA provides an extension to the depicted CICS integration capabilities.

5.3 CICS outbound request architecture

Figure 5-4 illustrates the CICS architecture that supports outbound access from CICS.

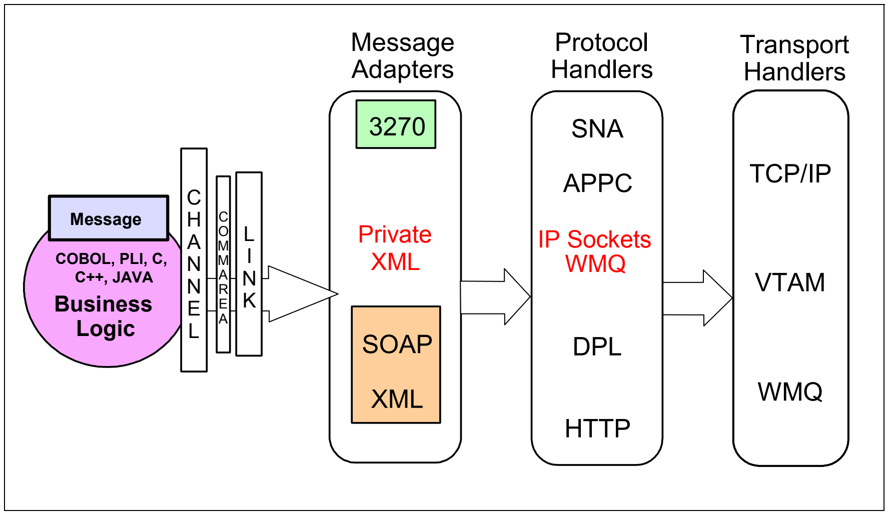


Figure 5-4 CICS outbound access architecture

Outbound access to external application interfaces requires the CICS client code to initiate the interaction. As discussed earlier, the business logic should not include access technology related code, so an adapter is typically invoked through an EXEC CICS LINK using a modern channel interface or a COMMAREA. The LINK extends the CICSPlex architecture choices and is a best practice to be used rather than a low-level language call.

CICS infrastructure services are grouped across three layers:

- Message adapters

  This is where the message serialization/deserialization occurs. It is transformed into the representation required by the external interface. For example, CICS supplies a SOAP/XML web services adapter.

- Protocol handlers

  This is where the protocol-related information required by the external interface is processed. CICS processes standard protocol implementations such as HTTP or DPL. TCP/IP sockets and WebSphere MQ require dedicated code.

- Transport handlers

  This is where the low-level transport is handled.

5.4 Adapters

An adapter addresses one or more of the following requirements:

- Business logic access technology transparency

  These fall into two categories:

  - A transport adapter that handles the low-level transport, such as TCP/IP
  - A protocol adapter that identifies the operation and supplies message envelope transparency

- Message serialization/deserialization transparency

  Message serialization/deserialization should be separate from the business logic. Such an adapter can also address EBCDIC data conversion. Typical examples are Java or XML to and from COBOL conversions.

- CICS LINK transparency

  When the existing interface is not using a CICS LINK interface (for example, a COBOL call) and the technology used implies a LINK interface (for example, a CICS Web services provider application), a simple LINK adapter is required that adapts a LINK interface into a COBOL call.

- Business service exposure with no modification to the existing programming interface

  This is a message mapping adapter that maps an existing CICS program interface to a new business interface. For example, an adapter may expose an interface of 20 XML elements that are mapped from the selected 20 fields in an existing COMMAREA that contains many more fields.
- Business service granularity

  A micro flow adapter can be used to increase the granularity of a service interface. It maps the course-grained business interface into multiple fine grained program interfaces.

The first three adapters focus on physical access technology interactions. They are mandatory. The last two adapters focus on abstracted business interactions. They are optional and address maturity issues.

CICS application adapters can be implemented on the client side (a JCA connector, for example) or the server side (for example, REST patterns) or both (for example, web services or WOLA). The complexity of the implementation depends on the access technology that is used. The use of application tooling can significantly simplify the solution.

Depending on the access technology, CICS adapters can take various forms:

- CICS TS native support
- Generated code from IDE tools
- Connector client code
- Bespoke user code
- A mix of implementations

Adapters can also be deployed as part of an Enterprise Service Bus (ESB) implementation, so that the message adaptation is performed by the ESB, either partially (message simplification for example) or fully.

5.4.1 Message serialization adapters

A primary function of a message adapter is serialization/deserialization that handles the mapping of a Java object or XML representation to a byte array. Such an adapter can be hand coded. However, this is a non-trivial task involving the calculation of field lengths and placement and data type conversions. The message serialization adapter implementation depends on the message representation technology, such as Java objects, XML, or JSON.

Java, Rational J2C tooling, and JZOS

The J2C wizards available with Rational Developer for System z (RDz) and Rational Application Developer (RAD), or with the JZOS Toolkit, provide tooling that generates code to handle message serialization/deserialization, producing getter and setter methods for each field in the message structure. The generated classes can then be used by Java methods to manipulate the COMMAREA or channel CONTAINERs as required.

The Rational J2C wizards can be used to create a complete CCI proxy with a few clicks. It not only generates the serialization/deserialization classes but also generates a skeleton J2C bean capable of accessing a CICS application. In a more industrial world, the few clicks can be replaced by batch procedures. The wizards also support optional EBCDIC data conversion. A typical CICS application interface access can be implemented and tested in a few steps:

1. Generate the serialization/deserialization data classes from the language structure.
2. Generate a skeleton J2C bean capable of accessing the CICS application.
3. Create a J2C bean method used for invoking the operation on the CICS server.
4. Create a web service to expose the functionality provided by the generated J2C bean.
5. Test the created web service using the web services explorer.

The JZOS Batch Toolkit for z/OS supplies a set of handy tools for Java on z/OS. Within these tools JZOS supplies a RecordClassGenerator utility for Assembler and COBOL language structures. This utility takes a sequential file as input and generates the serialization/deserialization Java bean class. The sequential file is generated from the ADATA compiler option. This is a two steps process:

1. Compile the CICS program with the ADATA compiler option to generate a binary file.
2. Run the JZOS Java utility to create the data classes from the binary file.

Pure XML or POX

This section addresses pure XML processing, where the XML message is not handled by the CICS Web service native support. XML message parsing and generation (deserialization/serialization) can be performed as follows:

- Using CICS TS V4 XMLTRANSFORM APIs

  You can use **`TRANSFORM XMLTODATA`** to convert XML to application data, and **`TRANSFORM DATATOXML`** to convert application data to XML. The DFHSC2LS and DFHLS2SC utilities generate the `xsdbind` file, which is analogous to the `wsbind` file created using the CICS Web services tooling.

  This is a nice balance between programming simplicity and infrastructure optimization.

- Using the COBOL XML PARSE verb

  This is a CICS neutral implementation. The programming style is more complex, as is the maintenance. The CPU consumption is higher than the CICS XMLTRANSFORM, but the processing can be executed partly on a zAAP processor.

- Using Java facilities

  This is a CICS neutral implementation. The Java code is likely to be generated from standard Java tooling. The CPU consumption is likely to be higher than using CICS XMLTRANSFORM, but the processing can be executed on a zAAP processor.

- Using the CICS DOCUMENT APIs for XML generation

  The DOCUMENT APIs can be used to create XML documents from predefined skeletons. The variable parts of the skeleton (identified by **`&myVariable;`** patterns) are substituted dynamically by the CICS DOCUMENT support. This is a low-cost and convenient way to generate error or information messages.

As a general rule, using the CICS XMLTRANSFORM APIs creates a nice balance between development agility and infrastructure costs.

Atom

With emerging Web 2.0 technologies, messages often need to be serialized as XML documents in the Atom Syndication Format. CICS supplies a native message serialization adapter which exposes CICS resources such as temporary storage queues and VSAM files as Atom feeds with no need for user-written code. An application program can also be exposed as a *feed*. In such a case, however, a user-written message adapter is required. This can be quite a complex task. The CICS Dynamic Scripting feature offers a simpler adapter implementation using WebSphere sMash or Project Zero facilities. The adapter is written in Groovy or PHP from a few lines of code using Project Zero renderers or APIs.

JSON

The JavaScript Object Notation (JSON) format is often used for serializing and transmitting structured data over a network connection. It is used primarily to transmit data between a server and web application in an AJAX web application programming model, serving as an alternative to XML. JSON message serialization can be implemented in CICS using the Dynamic Scripting feature. The message adapter is written in Groovy or PHP using Project Zero renderers or APIs.

5.4.2 Adapter and technology

Below is a summary of the different adapter implementations typically used in a CICS environment:

- Web service provider

  The CICS native web services support supplies adapters for the direct exposure of existing LINK interfaces. Simple message optimizations can be performed, for example, whitespace removal and specific field selection. The adapters are created using the DFHLS2WS or DFHWS2LS utilities and enabled by deploying the generated `wsbind` files. When required, a message adapter can be generated using the RDz Enterprise Service Toolkit (EST) or they can also be user-written.

- Web service requester

  The CICS native web services support supplies a full set of adapters for “simple” messages. The adapters are created using the DFHWS2LS utility and enabled by deploying the generated `wsbind` files. WSDL optimizations can be performed to simplify the messages with no impact on the service provider. An alternative adapter might be required for more complex messages. It can be generated using the RDz EST wizards or implemented by user-written code.

- JCA or ECI with the CICS TG for z/OS

  CICS TG and CICS TS supply a native transport adapter. The protocol and message serialization adapters are implemented on the client side. For JCA, the adapters are generated using Rational J2C wizards. For Java on z/OS, the JZOS Toolkit can be used to generate the message adapter.

- JCA with WOLA

  The transport adapter is supplied by the WOLA connector. The protocol and message serialization adapters are implemented on the client side. Rational

J2C wizards or the JZOS Toolkit can be used to generate the message adapter Java beans.

- CICS WOLA requester

  The transport and protocol adapters are hand coded from low-level WOLA connector APIs. The message serialization adapter is typically implemented on the WAS z/OS server side. Rational J2C wizards or the JZOS Toolkit generate the message adapter Java beans.

- CICS HTTP server

  CICS TS supplies a native transport adapter. The protocol adapter is hand coded using high-level CICS web support APIs, typically a WEB RECEIVE/SEND sequence. The message format is optional, and a user-written message adapter is normally required.

- CICS HTTP client

  CICS TS supplies a native transport adapter. The protocol adapter is hand coded from high-level CICS web support APIs, typically a WEB OPEN/CONVERSE/CLOSE sequence. The message format is optional and a user-written message adapter is normally required.

- REST server

  The Project Zero engine supplies a native REST support. JCICS Java classes supply a LINK interface to invoke the program resource identified by the REST URI. A REST adapter implements protocol, message serialization, and link adapter tasks from a few scripting language lines. It is composed of two parts:

  - A Project Zero REST event handler along with a JSON or Atom or XML message handler based on a few lines of PHP or Groovy scripting code.
  - A JCICS LINK client along with Java message serialization/deserialization. JZOS or Rational J2C wizards generate the message adapter Java beans.

- REST client

  A REST client is a specific use case of a CICS HTTP client.

- WebSphere MQ

  The CICS DPL bridge supplies native support for the transport and protocol adapters. The message adapter is likely to be a client implementation.

  When the DPL bridge is not used, a protocol adapter is required, typically a simple MQGET/MQPUT sequence. The message format is optional and a user-written message adapter is normally required.

- CICS Sockets

  Everything is hand coded.

The following sections of this chapter provide more details for each of the access technologies summarized in the above list.

5.5 CICS Web services

CICS Web services supports inbound and outbound connectivity to and from CICS applications. The following web service development approaches can all be used with CICS:

- Bottom-up development

  Where we want to expose an existing application as a web service, we would most likely consider a bottom-up approach. We start with the language structures for our current application and go through a process where we work upward, developing the WSDL and other infrastructure elements required for exposing a CICS application as a web service.

- Top-down development

  Where we wish to access an existing web service, we consider a top-down approach. We start with the WSDL, as published by the web service, and work downward, generating the CICS language structures, then developing the application code in order to create the CICS Web service application.

- Meet-in-the-middle development

  It is also possible that there is an existing web service definition that we are eager to use, and also an existing CICS application that can be used as the web service implementation. In this case, we can use our XML to language structure mapping tools to map the web service interface, to the CICS application program. This is called the meet-in-the-middle approach because the existing web service definition “meets” or “maps” to the original language structure interface. This approach requires a message mapping adapter.

Table 5-1 summarizes how the different approaches can be used in different situations.

Table 5-1 Web service development approaches

Approach	**Application**	**WSDL**	**Web service type**
Bottom-up	Existing	New	Service provider
Top-down	New	Existing	Service provider
Top-down	New	New	Service provider
Top-down	New	Existing	Service requester

Approach	Application	WSDL	Web service type
Meet-in-the-middle	Existing	Existing	Service provider

In this section, we focus on the application interface considerations for CICS Web services. For a full description of the different development approaches and development tools see the ITSO Redbooks publication *Application Development for CICS Web Services*, SG24-7126.

5.5.1 Transport and protocol adapters

The CICS Web services infrastructure provides transparent support for the transport and protocol adapters for SOAP over HTTP and SOAP over WebSphere MQ. The creation and deployment of the adapters are normally handled by the systems programmer.

5.5.2 Operation identification

The operation is defined within the WSDL file by the service provider. The WSDL operation name abstracts the real implementation name (that is, JEE method or CICS program). Interoperability is achieved using the first XML tag of the SOAP body as the representation of the operation.

Provider

When CICS is the service provider, the operation is typically the name of the target CICS program. The DFHLS2WS utility (used in the bottom-up approach) appends the "Operation" suffix to the CICS program name to build the operation name. For example **`MYPROG`** becomes **`MYPROGOperation`**. This default behavior can be modified if required. The DFHWS2LS utility (used in the top-down approach) uses the WSDL operation name.

Requester

A CICS Web service requester application must specify the WSDL operation name on the EXEC CICS INVOKE interaction:

```
INVOKE SERVICE(webservice name) OPERATION(operation name)
```

Note: Prior to CICS TS V4 the INVOKE WEBSERVICE call is used.

5.5.3 Message adapters

The CICS Web services infrastructure provides transparent support for message adapters. When you use the web services assistants, or other tooling, you do not have to write your own code for parsing inbound messages and for constructing outbound messages. CICS maps data between the body of a SOAP message and the application program's data structure. In addition, you can customize the way in which variable-length values and white space are handled by using settings on the CICS assistants and by adding facets directly into the XML schema.

Web service provider

Below we look at the specific considerations related to service provider implementations:

- ► Message serialization is typically performed transparently by CICS from the `wsbind` file generated by the DFHLS2WS utility or RDz. DFHLS2WS imposes several restrictions, such as the use of **`COBOL OCCURS DEPENDING ON`** and **`COBOL REDEFINE`**, which are not supported.

  Most restrictions can be addressed by using RDz to generate a message adapter. As a last resort, a user-written message adapter can be used to handle the most complex cases.

- ► Message mapping can be performed by simple modifications to the input language structure supplied to DFHLS2WS. Here are some common examples:

  – An existing COBOL field does not need to be exposed.

    It can be defined as a COBOL filler. DFHLS2WS will not expose it in the WSDL.

  – The name of an existing field is meaningless or too wordy.

    The field can be renamed safely. This is an easy answer to typical XML questions such as these:

    - Must XML tags be meaningful or human readable?
    - What is the impact of wordy XML tags on the infrastructure costs?

  A simple mapping adapter can be used to address other requirements, for example:

  – When an existing interface is using a COMMAREA message that is common to both the request and the response, DFHLS2WS can be used to generate a channel interface, and user-written code is used to move the channel CONTAINER fields to the COMMAREA.

  – When an existing application interface uses a COBOL CALL rather than an EXEC CICS LINK, the need to create a link adapter is a good opportunity to optimize the message interface, so it is delegated to a message mapping adapter.

  - When a SOAP-FAULT message is to be returned rather than an error code in the response message.
  - When an existing program interface uses a COMMAREA, the business message is longer than 32 kb, and the current web services implementation segments/rebuilds the message using an intermediate server. A better solution might be a direct exposure of the business message from a CICS channel, which would lower the number of client/server interactions and simplify the client proxy implementation.

Meet-in-the-middle approach

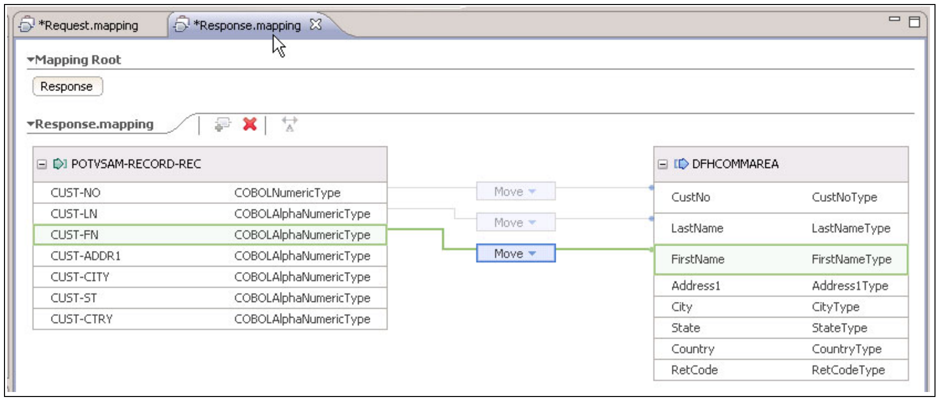
A simple *submit and go* or *click and go* approach using DFHLS2WS imposes a language structure coupling to the exposed interface. The generated XML schema might not always be appropriate for the service requester, especially if the requester is a business partner rather than an internal requester. These are common examples:

- When the use of COBOL `PIC X(6)` imposes a mandatory XML string element of a fixed length (six characters in this case).
- When using a COBOL `OCCURS` 'list' definition, DFHLS2WS generates the equivalent fixed list XML element (minOccurs=maxOccurs=COBOL OCCURS value). Most of the time, this is likely to be acceptable. But for long lists or nested lists, or high throughput workloads, this can be an issue and might require being optimized.

To address such situations, a meet-in-the-middle approach can be used to create a user message mapping adapter:

- Use the bottom-up DFHLS2WS utility to generate the WSDL and XML schema from the language structure.
- Use a WSDL graphical tool (for example, RDz) to modify the XML schema and to transform it into a more suitable service interface.
- Use the top-down DFHWS2LS utility to generate the new language structure.
- Write a message mapping adapter to map the new interface to the existing CICS program interface.

This procedure can be used for simple mapping requirements. However, you may prefer to use the RDz Enterprise Service Toolkit (EST) wizards to generate the mapping adapter. Figure 5-5 shows a screenshot from the RDz graphical mapping wizard.



POTVSAM-RECORD-REC	
CUST-NO	COBOLNumericType
CUST-LN	COBOLAlphaNumericType
CUST-FN	COBOLAlphaNumericType
CUST-ADDR1	COBOLAlphaNumericType
CUST-CITY	COBOLAlphaNumericType
CUST-ST	COBOLAlphaNumericType
CUST-CTRY	COBOLAlphaNumericType

DFHCOMMAREA	
CustNo	CustNoType
LastName	LastNameType
FirstName	FirstNameType
Address1	Address1Type
City	CityType
State	StateType
Country	CountryType
RetCode	RetCodeType

Figure 5-5 RDz meet in the middle data mapping wizard

When new CICS applications have to be exposed, the best practice is to model the WSDL from a business service perspective, and then to generate to a language structure interface from DFHWS2LS.

Web service requester

Below we look at the specific considerations related to CICS service requester implementations. While a CICS Web service provider application can be invoked transparently by the CICS infrastructure, a CICS Web service requester program explicitly uses **`EXEC CICS INVOKE SERVICE`** to invoke the service provider interface. This is normally done in a new message adapter program:

- Simple message serialization can be performed transparently by CICS from the `wsbind` file generated by the DFHWS2LS utility or RDz.

  In real life, however, most of the CICS Web service requester implementations access an object-oriented implementation of a web service provider. The object implementations imply intensive use of XML facets, such as nillable or minOccurs=0 or maxOccurs=unbound, which are not natively supported by procedural languages. A hand-coded message serialization adapter is required most of the time. DFHWS2LS provides a set of handy transformation capabilities in order to simplify the adapter implementation. Here are a few typical examples:

  – An XML boolean element is exposed as a single byte field whose content is X’00’ when false and X’01’ when true.

  - A nillable XML element attribute is exposed as a single byte field whose content is X'01' when nill.
  - A minOccurs=x and maxOccurs=y XML facet conversion depends on the INLINE-MAXOCCURS=a_limit parameter of the DFHWS2LS utility.

    IF INLINE-MAXOCCURS is less than or equal to the maxOccurs value, then it is converted *inline* into a counter field and a fixed array interface. The counter is the number of occurrences of the field element in the array. Otherwise a *container-based* conversion occurs, which uses a counter field and a 16-character CICS CONTAINER name field interface. The named container contains the list elements.

    The best practice is to use the inline conversion for small lists and to use the container-based mapping for larger amounts of data. The container-based mapping also optimizes memory consumption so long as the SET option of the GET CONTAINER is used.

- Message mapping is normally required when a CICS service requester application invokes a web service that has an object-oriented implementation. Whereas language structure descriptions imply a tight message coupling, with object oriented implementations it is the exact opposite: Everything is optional, nillable, unbound, and so on. For interoperability or service orientation, such a "help yourself interface" is not always a best practice. A significant part of the ROI is related to the optimization of the service interfaces described in the WSDL. The WSDL file should be an enabler to automation, reuse, first time successful interactions, and predictable execution behavior.

  From real-life experience there are two common situations:

  - Internal local service provider

    In such a case the service interface is generally optimized by the service provider team using WSDL and XML tooling.

  - External remote service provider (for example, a business partner)

    In such a case the service interface can be optimized locally or using an ESB. The local optimization is performed using WSDL tooling to add restrictions to the XML schema elements. The WSDL describes the service contract and, while it cannot be extended by the consumer, its scope can be restricted by the consumer. This is where the message adapter logic can be simplified and optimized, for example, request message mandatory elements can be described as such, and sensible limits can be defined with no impact on the provider.

    Complex interfaces result in complex message conversion logic. In such cases an ESB might be a better solution. For other situations, a pragmatic approach might be a combination of message mapping in CICS and in an ESB.

5.5.4 XML validation

By default the CICS Web services pipeline does a simple message validation. For example, unknown tags are rejected. Full XML message validation can be activated using the VALIDATE option of the WEBSERVICE definition.

Note: Validation of a SOAP message against a schema incurs considerable processing overhead, and you should normally specify VALIDATION(NO) in a production environment.

5.5.5 Binary or invalid XML messages

CICS supports and controls the handling of MTOM messages in both web service provider and requester pipelines using an MTOM handler program and XOP processing.

Where MTOM/XOP is not the right solution, and when specific message processing is required, the CICS message data mapping can be disabled by using the XML-ONLY parameter of DFHWS2LS.

5.5.6 Message exchange pattern

The WSDL 2.0 specification supports the explicit definition of Message Exchange Patterns (MEPs) within the WSDL document. Four MEPs can be used with CICS Web services:

- In-only, where no response is to be returned.
- In-out, which is an RPC equivalent.
- In-optional-out, which is a possible combination of the two previous patterns. It adds some complexity to the interface, but it is supported.
- Robust-in-only, where a response message is returned only if it is a SOAP fault.

The WSDL 1.0 specification does not support MEPs explicitly. However, CICS supports the following patterns implicitly:

- In-out: The WSDL defines request and response messages for an operation.
- In-only: The WSDL does not define a response message for an operation

5.5.7 Data conversion

Web services messages are encoded in UNICODE. The CICS infrastructure automatically handles EBCDIC data conversion. The LOCALCCSID parameter of the CICS SIT defines the local EBCDIC codepage to be used, for example, 1147 for France.

5.5.8 Coupling considerations

Web services technologies are often referred to as being loosely coupled. This is true from an IT implementation point of view:

- Loosely coupled client/server transport and protocol
- Loose coupling between the WSDL artefacts, the application interface, and the CICS implementation
- Loosely coupled Unicode exchanges

From an application interface point of view, however, the degree of coupling is dependent on the answer to the question “What happens if I modify the format of a message?” If the answer is that for any modification “I must supply a new WSDL to all my consumers,” then this cannot be considered a loosely coupled solution.

The coupling of the message is to be balanced with interoperability of the interactions. This is where simple message adapters and ESB-based solutions can be useful. Also, CICS supports the xsd:any and xsd:anyType XML types, which can be used to extend an interface with no impact on existing users.

5.6 CICS TG for z/OS

A CICS TG for z/OS connector access to a CICS application interface is a straight DPL call issued from the CICS mirror transaction to this interface. It supports both the COMMAREA and the channel message representation. CICS TG provides an inbound-only access to CICS.

CICS TG supports the JCA CCI interface and since CICS TG V8 it also supports ECIv2 calls from non-Java clients.

5.6.1 CCI programming model

Applications using the CCI have a common structure, independent of the EIS that is being used. The JCA defines connections and ConnectionFactories, which represent the connection to the EIS.

An application must start by obtaining a ConnectionFactory from which a connection can be obtained. The properties of this connection can be overridden by a ConnectionSpec object. The ConnectionSpec class is the CICS-specific class ECIConnectionSpec.

After a connection has been obtained, an *interaction* can be created from the Connection to make a particular request. As with the connection, interactions can have custom properties set by the CICS-specific InteractionSpec class (ECIInteractionSpec). To perform the interaction, you call its execute() method and use CICS Record objects to pass the message data (Example 5-1).

Example 5-1 Common Client Interface

```
ConnectionFactory cf=<JNDI lookup>
Connection c = cf.getConnection(ConnectionSpec)
Interaction i = c.createInteraction()
InteractionSpec is = newInteractionSpec();
i.execute(spec, input, output)
```

The ConnectionFactory has custom properties that are used to specify connection details, for example, the Gateway daemon to be used is set using the ConnectionURL property. When the ConnectionFactory has been created, it can be made available for use by any enterprise application through the JNDI.

5.6.2 Transport and protocol adapters

For the CICS TS application, the CICS TG for z/OS connector provides transparent support for both the transport and protocol adapters.

For the CICS TG client application, the transport and protocol adapters are provided through the CCI interface or the ECIv2 APIs supplied by the CICS TG connector.

5.6.3 Operation identification

The operation is the name of the CICS program name that is linked to. It is supplied by the CICS TG client:

- For CCI, the setFunctionName method of the ECIInteractionSpec class is used to set the program name property.
- For ECIv2, the ECI program name is used to specify the CICS program name.

5.6.4 Message adapters

A COMMAREA or a channel message is represented as a Java object or an ECIv2 byte array. The serialization adapter is typically implemented on the Java side using RDz or RAD J2C tooling. When the JEE container is WebSphere Application Server for z/OS, the JZOS Toolkit can also be used. For ECIv2 clients, this task is not required.

Because the COMMAREA is common to both the request and the response messages, it is common practice to set the length to a maximum value (normally the maximum length of the response message). This can be a best practice in terms of coupling or versioning. To optimize the network flows, the CICS TG implements a mechanism known as COMMAREA null stripping, which strips trailing nulls from the COMMAREA message before transmitting it across the network. Null stripping applies to both the request and the response messages and is transparent to the client application programs and CICS server programs, which always see the full-size COMMAREA.

This is a direct COMMAREA or channel message exposure. For such a bottom-up approach, a user-written CICS message mapping adapter is not normally required.

5.6.5 Message exchange pattern

The CCI interaction supports the SYNC_SEND, SYNC_RECEIVE, and commonly used interaction SYNC_SEND_RECEIVE:

- SYNC_SEND_RECEIVE is a synchronous call.
- SYNC_SEND and SYNC_RECEIVE supply a form of an asynchronous interaction where the client code polls for the response. The interactions require the CICS server connection to be active.

ECIv2 supports the ECI_SYNC pattern, which is functionally equivalent to CCI SYNC_SEND_RECEIVE.

The pattern used by the client is transparent to the target CICS application.

5.6.6 External Call Interface (ECI)

The ECI emulates an EXEC CICS LINK DPL call for non CICS TS clients. This means that the CCI or ECIv2 client code supports the properties that we described earlier for the LINK interface.

In a CCI-managed environment most of the connection properties, such as the CICS server name, are set using a connection factory and are transparent to the client interface. The ECIConnectionSpec object allows the client application to override certain properties. For example, it can set the UserName and password properties for the connection.

The ECIInteractionspec object contains the details of the CICS application interface to be accessed. Within a rich set of properties here are the most important:

- The interaction verb (for example SYNC_SEND_RECEIVE)
- The function name (the CICS program to LINK to)
- The Java record representation of a COMMAREA
- The ECIchannelRecord representation of a channel
- The CICS transaction code to be used to identify the workload to be executed within the CICS server (set by using the setTPNName() method of the ECIInteractionSpec object)

A best practice is to set a specific CICS transaction code for an interaction. Using specific transaction codes enables CICS infrastructure optimizations such as workload management, security, and accounting. Another good practice is to use the CICS TG for z/OS support for mapping a server logical name to a CICS physical name, thus hiding the physical name from the client implementation.

5.6.7 EBCDIC data conversion

COMMAREA message data conversion in CICS requires a DFHCNV program entry. For character containers, channel messages are converted transparently by CICS on the GET CONTAINER calls.

When complex messages are exchanged, RDz or RAD J2C tooling can be used to generate the data conversion code, for example, when using COBOL zoned decimal fields with a non-English codepage. In this case no entry must be defined in DFHCNV, and the conversion is performed on the Java client side.

5.6.8 Coupling considerations

Using the CICS TG connector to access a CICS application interface implies a medium coupling:

- The message is a byte array, typically a COMMAREA. A channel message interface reduces this coupling.
- CICS infrastructure properties such as the program name or the transaction code are set by the client code. The use of a framework lowers the coupling but it does not suppress it.
- It is a direct COMMAREA or channel exposure. The use of J2C tooling to create record beans lowers this coupling.
- The EBCDIC data conversion requires application programmer and system programmer synchronization. Character data messages lowers this coupling.

5.7 WOLA

WOLA provides inbound and outbound connectivity between CICS and WebSphere Application Server for z/OS:

- Inbound connectivity to CICS from a servlet or EJB is similar to the CICS TG for z/OS connector support, in that is based on the Common Client Interface (CCI) and implements ECI-like interactions. This is a direct LINK interface exposure that supports both the COMMAREA and the channel message representation.
- Outbound connectivity from a CICS application requires some coding to the WOLA-specific APIs.

5.7.1 Transport and protocol adapters

The transport is an optimized cross-memory pipe between CICS TS and WebSphere Application Server for z/OS that is used for passing data (normally byte arrays) between applications very quickly. Transport is handled by the WOLA components:

- For CICS LINK interface server applications, the WOLA components provide a transparent support for the transport and protocol adapters. The Link Server Task accepts calls from WebSphere Application Server for z/OS, then issues EXEC CICS LINK against the named CICS program, passing either a COMMAREA or a channel.
- The Link Server Task does not assist in calls outbound from CICS. Instead, the CICS application initiates the interaction using the WOLA APIs.

There are two broad approaches:

  – Embed the WOLA API processing in the CICS program itself.
  – Write a custom WOLA bridge program that can be linked to by the existing business logic programs. In this sense it becomes a kind of *WOLA Service* to existing CICS programs to utilize as your needs require.

5.7.2 Operation identification

For WebSphere client applications, the operation is the real CICS program name. It is set using the InteractionSpecImpl serviceName property.

For CICS client applications the operation is the EJB Home JNDI name.

5.7.3 Message adapters

WOLA itself pays no attention to the contents of what is being passed back and forth between WebSphere Application Server and CICS. It does not care about data layout and it does not care about code pages:

- For inbound connectivity to CICS, the message serialization adapter is implemented on the Java side using RDz or RAD J2C tooling, or the JZOS Toolkit.
- For outbound connectivity from CICS, the message is a byte array, the 31-bit address of which is passed on the WOLA API calls. The BBOA1INV API provides a simple invoke call where the maximum length of the response must be known. The BBOA1GET API is a receive call that supports limited data truncation support.

5.7.4 Message exchange patterns

WOLA is primarily a synchronous connector, but it does have limited assynchronous support:

- For inbound connectivity to CICS, the WOLA CCI connector supports the SYNC_SEND_RECEIVE pattern.
- For outbound connectivity from CICS, you can use either the basic WOLA APIs or the advanced APIs:
  - The basic APIs provide ease of use but limit the flexibility of operations. Specifically, they assume synchronous control. For example, on the BBOA1INV (invoke) API program control is not returned to the CICS program until WAS returns the response. The CICS task and the WOLA connection are tied up during that time.
  - The advanced APIs allow you to operate asynchronously. That allows your program to receive program control immediately, which allows your program to go off and do other work. This allows for greater utilization of resources.

5.7.5 EBCDIC data conversion

EBCDIC codepage conversion can be done within CICS or within WebSphere Application Server:

- For inbound connectivity to CICS, codepage conversion is normally done using record beans generated by RDz or RAD J2C tooling.

  Note: The CICS data conversion program (DFHCCNV) is not called from the LINK Server Task.

  For character containers, channel messages can be converted transparently by CICS on the GET CONTAINER calls.
- For outbound connectivity from CICS, codepage conversion is normally done in WebSphere Application Server using record beans generated by RDz or RAD J2C tooling.

5.7.6 Coupling considerations

Using WOLA implies a tight coupling, especially when the WOLA APIs are used. This is to be expected for such a high-performance technical connector:

- The WOLA connector can only be used for connectivity between CICS and WebSphere Application for z/OS.
- The WOLA APIs are low level and require specific programming skills.
- CICS infrastructure properties, such as the program name or the transaction code, are set by the client code. The use of a framework lowers the coupling, while it does not suppress it.
- The message is a byte array. For inbound calls to CICS this is typically a COMMAREA, although a channel message interface reduces this coupling. Inbound requests run under the same link server transaction code.

5.8 CICS web support

The HTTP specification defines the client/server exchange formats to be used to communicate between an HTTP client and an HTTP server:

- HTTP request

  An HTTP request is made by a client, to a named host, which is located on a server. The aim of the request is to access a resource on the server.
- HTTP response

  An HTTP response is made by a server to the client. The aim of the response is to provide the client with the resource that it requested or to inform the client that the action that it requested has been carried out, or to inform the client that an error occurred in processing its request.

An HTTP message is composed of a request line, including such things as the method, URL path and the status code, a series of HTTP headers, and an optional body.

CICS supports the HTTP 1.0 and HTTP 1.1 specifications, and the web support can be used for inbound and outbound requests.

5.8.1 Transport and protocol adapters

When using the CICS web support, you will most likely write an HTTP-aware transport and protocol adapter. CICS web support supplies a rich set of APIs (EXEC CICS WEB), which hides the complexity of HTTP from the adapter programmer.

Simple interactions that do not need access to specific HTTP information, such as header content or query string data, do not require any transport-specific code. More complex interactions might require some knowledge of the HTTP information. CICS web support provides specific HTTP transport APIs that can be used to simplify the task of writing an HTTP-aware adapter. This support includes these:

- HTTP header access, including read, write, and discovery of headers
- URL parsing
- Query string data handling
- Manipulation of HTTP context information such as the method (GET, for example), hostname, scheme, version, path, or TCP/IP port number.

HTTP body

The HTTP body contains the message. CICS web support supplies the WEB RECEIVE and the WEB SEND APIs, which can be used to process HTTP messages. Both APIs support a message buffer or a CONTAINER interface:

- WEB RECEIVE

  The message buffer API supports truncated messages. Truncation occurs when a received message is longer than expected. Application interfaces should always define whether fixed length or variable messages are exchanged. When variable messages are exchanged, a maximum size should always be defined. When it is possible to receive very long messages, an average message size can be defined. These different patterns determine the truncation protocol to used by CICS.

  The CONTAINER API does not define a maximum message length.

  Note: Unless the CONTAINER is to be passed to other channel interfaces, the buffer API is normally the best practice.

- WEB SEND

  If query string data information needs to be added in the URL, it must be specified using the QUERYSTRING parameter of the WEB SEND API.

CICS web support supports HTTP 1.1 chunking, which is how HTTP supports message segmentation. For inbound messages, CICS handles the chunking sequences. For outbound messages the WEB SEND API must define the chunk segment length.

When CICS is the client, it needs to connect to the HTTP server. The connection is established by a WEB OPEN API call and closed (or returned to the pool) by a

WEB CLOSE API call. The URIMAP parameter hides the physical server name from the application.

HTTP-specific information

The HTTP protocol addresses interoperability issues such as message format identification and error code representation by using a set of protocol parameters, including these:

- Mediatype

  The HTTP mediatype information must be supplied on a WEB SEND (for example, text/xml) according to the message format to be included in the HTTP body.

- Method

  An HTTP request message must contain a valid HTTP method. When CICS is a client, the WEB SEND supports the GET, POST, PUT, DELETE, HEAD, OPTIONS, and TRACE methods. When CICS is the server, the method can be retrieved by a WEB EXTRACT.

- Status code

  An HTTP response message must contain a valid HTTP status code (for example, HTTP 500). It must be set on a WEB SEND when CICS is a server. It is retrieved on a WEB RECEIVE when CICS is a client.

5.8.2 Operation identification

The operation of an HTTP request is identified by the Unified Resource Identifier (URI), which is the path part of the URL. In CICS the URIMAP resource defines the characteristics of the application interface such as these:

- The program name
- The transaction code
- Any static information to be returned (for example, a friendly error message to be returned in case of temporary service unavailability)
- HTTP server redirection information

For a CICS client, the URIMAP on the WEB SEND API specifies the operation (path information) for the specific resource in the server that the CICS application will access.

5.8.3 Message adapters

The message format is defined in the Content-Type HTTP header information and can be character or XML or other formats. A message mapping adapter is optional.

5.8.4 Message exchange pattern

The HTTP protocol does not define a message exchange pattern. HTTP interactions are typically SEND_RECEIVE exchanges.

CICS supports HTTP 1.1 pipelining, which allows a client to issue multiple requests without waiting for a response. The responses are returned by the server in the sequence in which the requests were received. This is a form of an asynchronous pattern.

5.8.5 EBCDIC data conversion

The WEB RECEIVE and WEB SEND API support EBCDIC character data conversion as required. This means that the mediatype header information defines the body as character data. When the message buffer API is used, the SRVCONVERT option indicates that data conversion is to take place. When the CONTAINER API is used, the conversion is performed on the GET CONTAINER call.

The default HTTP body codepage is defined by the Content-Type HTTP header. CICS uses it as the client codepage. It can be overridden.

The default EBCDIC codepage is defined by the LOCALCCSID option in the CICS SIT. It can also be overridden.

5.8.6 Coupling considerations

CICS web support is seen as a medium coupled access technology. In fact, it has loose coupling characteristics in addition to tight coupling characteristics.

It can be seen as a loosely coupled solution in the following ways:

- The HTTP protocol is widely implemented.
- The URL abstracts the operation.
- HTTP standard headers solve the Content-Type or Mediatype or Content-Length coupling issues.
- The URIMAP decouples the adapter from concrete implementation details.

It can also be seen as a tightly coupled solution in the following ways:

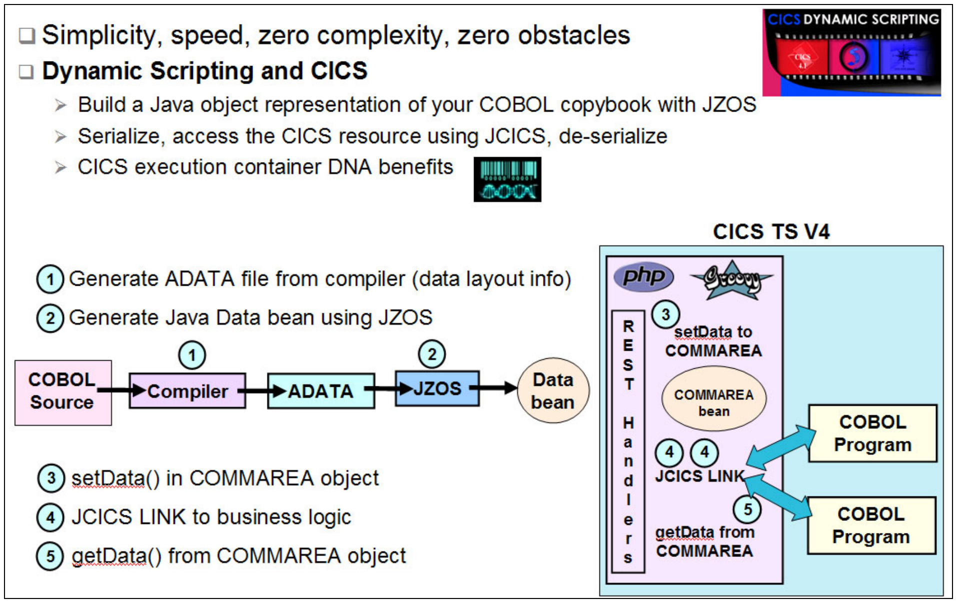
- Message serialization/deserialization is not defined, and the level of coupling depends on the technology used, from tight byte array messages to loose xml messages.
- The HTTP method usage pattern is not defined. REST patterns help to address this.
- Adapters must be hand coded.

5.8.7 REST and dynamic scripting

An important concept in REST is the existence of resources (sources of specific information), each of which can be referred to using a global identifier (a URI). To manipulate these resources, components of the network (clients and servers) communicate through a standardized interface (using HTTP and HTTP verbs) and exchange representations of these resources (the actual documents conveying the information).

REST defines a set of conventions to give a meaning to URIs and HTTP methods. For example, an HTTP GET on a collection URI is a LIST request, whereas a GET on a resource URI is a RETRIEVE request. CICS web support supplies the native HTTP support, but specific application code is required to implement REST-style applications.

The CICS Dynamic Scripting feature supplies a native REST support using the Project Zero event driven support. The ITSO Redbooks publication *Introduction to CICS Dynamic Scripting*, SG24-7924, provides detailed information about implementing the feature. When using the feature, a REST adapter can be implemented using a few lines of Groovy or PHP. Figure 5-6 describes the adapter implementation.



- ❑ Simplicity, speed, zero complexity, zero obstacles
- ❑ **Dynamic Scripting and CICS**
  - ➢ Build a Java object representation of your COBOL copybook with JZOS
  - ➢ Serialize, access the CICS resource using JCICS, de-serialize
  - ➢ CICS execution container DNA benefits

1. Generate ADATA file from compiler (data layout info)
2. Generate Java Data bean using JZOS
3. setData() in COMMAREA object
4. JCICS LINK to business logic
5. getData() from COMMAREA object

Figure 5-6 A REST adapter

The adapter leverages the Project Zero event-driven support to handle the REST-style interaction patterns. CICS Dynamic Scripts run within a CICS JVM. This enables the use of JCICS classes to access CICS ressources. The JCICS LINK interface supports the COMMAREA or the CHANNEL interface. These are the steps required (Figure 5-6):

1. The COBOL message representation is compiled with the ADATA COBOL compiler option. This generates a binary adata file.
2. The JZOS utility generates the message access Java bean and the geter and setter methods from the adata file.
3. At run time the PHP or Groovy script creates a COMMAREA object from the bean setter methods.
4. The dynamic script issues a JCICS interaction.

Note: In Figure 5-6 the REST adapter exposes a coarse-grained message from two medium-grained COMMAREAs.

5. The script receives the response COMMAREA using the bean getter methods.

5.9 WebSphere MQ

A WebSphere MQ interface can be used by CICS applications in two ways:

- An application can use the WMQ APIs for getting or putting messages to queues. The best practice is to use an adapter so that the business logic is encapsulated from the WMQ transport.
- The WebSphere MQ DPL bridge for CICS provides an alternative option that allows DPL interactions inbound to CICS COMMAREA-based programs. No adapter is required in this case.

The ideal interaction use case for WebSphere MQ is an asynchronous model. However, in real life it tends to be a used as a synchronous RPC style interaction.

5.9.1 Transport and protocol adapters

The CICS WebSphere MQ attachment enables the use of the MQ APIs within CICS applications. CICS connects to a queue manager and provides transparent transport adapter support.

WMQ API enabled applications require a user-written protocol adapter, which typically uses an MQGET/MQPUT interaction sequence.

CICS DPL programs that are accessed through the WebSphere MQ DPL bridge benefit from the protocol adapter supplied by the bridge. While the bridge hides the WMQ details from the CICS application, it might expose the CICS LINK properties to the WMQ client application via the MQCIH message header.

5.9.2 Operation identification

The operation is likely to be identified by the queue name. When this is not the case, for example, when the queue serves multi-purpose messages, the operation must be identified somewhere in the message descriptor or in an application-specific header.

The CICS DPL bridge implementation uses such a multi-purpose message queue. The operation is the CICS program name and is identified by the first eight characters of the message or by a field in the MQCIH message header.

5.9.3 Message adapters

A message adapter uses WMQ native APIs to receive the message, transform it if required, and call the CICS business logic program. A reply message can be sent using the reply-to queue defined in the message.

The message can be character, bytes, or XML, as defined in the WMQ message descriptor. When message serialization is required, for example, when the client is a Java application, it is normally implemented on the client side. Java implementations benefit from the Rational J2C tooling or JZOS Toolkit.

When using the CICS DPL bridge to directly access a COMMAREA-based appplication, a message mapping adapter is not normally required.

5.9.4 Message exchange pattern

Most of the time, the message exchange pattern is not defined explicitly. Although the MQMD header contains information that might imply the type of interaction, such as the message type.

WebSphere MQ is an asynchronous transport mechanism. When using a synchronous application model (MQPUT followed by MQGET) care needs to be taken to manage timeout situations.

Event-driven processing and publish/subscribe are alternative patterns in which CICS can participate.

5.9.5 The MQ DPL bridge client interface

The client sets a number of fields in the MQMD and MQCIH structures in request messages for the CICS DPL bridge.

The WMQ DPL bridge interface operates in different ways:

- If a message has a format of anything other than MQCICS, then the simple DPL bridge is assumed. The first eight characters of the message body is used as the target of the CICS LINK, with the remainder of the message body passed as a COMMAREA.
- If the message format is MQCICS, then the message has a CICS header prior to the message body. This header contains control information that can take these actions:
  - Direct multiple transactions to be executed.
  - Direct additional, or alternative, authorization.
  - Provide a different message type.
  - Provide other directives to the bridge-processing program.

5.9.6 EBCDIC data conversion

WebSphere MQ supplies native message data conversion. The MQMD supplies the client encoding information in addition to the format information.

Conversion is carried out by two different routines, one for the MQCIH structure and another for the data or vectors supplied in the message. You can ensure that the MQCIH is converted by specifying `MQFMT_CICS` in the MQMD.Format field.

If you are driving a DPL program that neither receives nor returns COMMAREA data, or if the COMMAREA data is purely character data, you can achieve data conversion by specifying `MQFMT_STRING` in the MQCIH.Format field. If your COMMAREA data is not purely character data, you must write your own conversion routine.

5.9.7 Coupling

WebSphere MQ offers a loose coupling between applications. However, potential coupling issues might need to be addressed:

- Message serialization/deserialization is not defined. The level of coupling depends on the technology used, from tight byte array messages to loosely coupled xml message.
- The message exchange pattern is not defined. Specific WMQ message headers can be used to identify the type of exchange pattern being used.
- Message adapters need to be user-written.

Note: A number of advances have been made with WMQ V7 to provide better linkages between request and reply messages, including message properties.

5.10 CICS sockets

Sockets implementations use a low-level technical interface and require specific programming skills. CICS sockets-enabled programs are normally early applications inherited from the past. New TCPI/IP connector applications should be based on the HTTP protocol which can be seen as a mature program to program TCP/IP sockets protocol.

The best practice is to use an adapter so that the business logic is encapsulated from the sockets transport. With CICS sockets implementations, all integration aspects must be addressed by user-written code. Issues to consider include these:

- Has all the data been received?

  TCP/IP sockets do not guaranty that all the data has been received. This requires a loop on the read operation and information about the real length of the message in order to know when to stop the loop.

- What happens if a socket call does not return control, for example, on a connect?

  A best practice is to use asynchronous (or non-blocking mode) calls rather than synchronous (or blocking) ones.

- How are hostnames handled?
- How is data conversion done?

  This can be complex, as the codepage information sent with a request might also need to be converted.

- How do I handle error situations?
- How do I produce meaningful error messages?
- What impact will IPv6 have on the application?

Some of these issues can be addressed by the use of a private header that is flowed with each message. Other issues need to be addressed by low-level programming techniques.

5.10.1 Transport and protocol adapters

When CICS is a server, the CICS sockets feature supplies a concurrent IP listener transaction (CSKL):

- CSKL can be implemented as a standard listener, where the initial send emitted by a client must supply a CICS header at the beginning of the message. This header supplies a mandatory CICS transaction code of the child server along with optional data.
- CSKL can also be implemented as an enhanced listener, where the CICS header is not required. The default CICS transaction code is set in the CSKL configuration file and can be overridden by an exit.

You can also create your own listener.

When CICS is a client, the sockets application must connect to the external sockets server using an IP address. The API supports gethost type calls.

Messages are exchanged using SEND and READ API variations. As discussed previously, the message receive code should implement a loop based on the message length information.

5.10.2 Operation identification

The operation identification is a private implementation.

5.10.3 Message adapters

This task is performed by user-written application code. A private header can be used to identify the message data type, for example, text/XML.

5.10.4 Message exchange patterns

No message exchange patterns are defined. Sockets programming requires the client and server implementations to be in sync in order to avoid situations in which they are in receive state at both sides.

5.10.5 EBCDIC data conversion

The CICS sockets feature supplies conversion routines limited to a single codepage, the defaults being EBCDIC 037 and ASCII ISO-8859, which limits the conversion to American English. To support other codepages, such as the French 1147 codepage, the provided translation tables must either be modified, which is a time-consuming task, or other conversion services must be used, such as the EXEC CICS CONTAINER API or native calls to the iconv conversion service.

5.10.6 Coupling considerations

CICS sockets implies a very tight coupling at every level.

6

Security

In a recent study done by IBM[1], the majority of CEOs and CIOs identified risk management as an area where they will focus IT to help their organizations' strategy over the next five years. Therefore, because CICS applications and their associated data constitute some of the most valuable assets owned by an enterprise, the protection of these assets is a essential part of any CICS integration project.

When you consider the security design for your CICS application, you need to weigh the following key issues:

- How will you ensure that the users of the application are properly authenticated?
- What authorization mechanisms will be used to protect access to the CICS system and access to resources such as transactions, files, and databases?
- How will you protect the confidentiality of data that is transported between the different tiers of the physical configuration?
- How will you meet audit and compliance regulations?

In this chapter, after a review of general security objectives, we discuss the main security considerations for each of the strategic CICS integration technologies.

[1] The Essential CIO, Insights from the Global Chief Information Officer Study, found at `http://www.ibm.com/services/c-suite/cio/study.html`.

6.1 Security objectives

Whereas business agility through better integration and a service-oriented approach are key to innovation and gaining market share, this cannot be done at the cost of customer confidentiality and transaction security. Measures taken to minimize risk are an important ingredient to any viable IT strategy. When our countermeasures are insufficient and are surmounted by either intentional or unintentional events, the cost to a business can be staggering.

In this section we discuss the key objectives in the creation of a secure infrastructure.

6.1.1 Measures required to secure the infrastructure

A complete security solution puts mechanisms in place to achieve the following objectives:

- Authentication

  Authentication is the process of validating the identity claimed by the accessing entity. Authentication is performed by verifying authentication information provided with the claimed *identity*. The authentication information is generally referred to as the accessor's *credentials*. A credential can be the accessor's name and password. It can also be a *token* provided by a trusted party, such as a Kerberos ticket or an X.509 certificate.

  Note: Authentication is usually one of the earliest steps in a request workflow. When authenticated, an identity can be *asserted* to the downstream process steps, meaning that these steps trust the upstream steps to have already successfully authenticated the identity.

- Identification

  Identification is the ability to assign an identity to the entity accessing the system. Typically the identity is used to control access to resources. Depending on the security model in which the identification is performed, the identity may come from the authentication credentials or it might be asserted from another server.

- Authorization

  Authorization is the process of checking whether an identity that has already been authenticated should be given access to a resource that it is requesting. A typical implementation of authorization is to pass to the access control mechanism a *security context* that contains the identity that has been authenticated.

- Integrity

  Integrity ensures that transmitted or stored information has not been altered in an unauthorized or accidental manner. Typically it is a mechanism to verify that what is received over a network is the same as what was sent.

- Confidentiality

  Confidentiality ensures that an unauthorized party cannot obtain the meaning of the transferred or stored data. Typically confidentiality is achieved by encrypting the data.

- Auditing

  With auditing, you capture and record security-related events (such as a user signing onto or off of a system) so that you can analyze them later, perhaps after a breach of your security has occurred.

- Non-repudiation

  Non-repudiation means that a sender and a receiver of data are able to provide legal proof to a third party that the sender did send the information and the that receiver received the identical information. Neither side is *able to deny*.

  A solution for non-repudiation must provide proof of the integrity and origin of data, and an authentication mechanism that with very high assurance can be claimed to be genuine. The most common method of asserting the origin of data is through the use of digital signatures and a form of Public Key Infrastructure (PKI).

  Note: Most of the CICS integration technologies described in this book support the use of digital signatures (SSL/TLS or XML digital signatures), which can be used as part of a solution for non-repudiation. This book, however, does not cover other aspects of non-repudiation solutions.

Different CICS integration projects will have different security objectives. After the specific objectives are understood, you can evaluate the types of security mechanisms that best meet the specific set of objectives.

6.1.2 Barriers to implementation

Although security is an essential part of any solution design, implementation can be hampered by a variety of technical factors. The following list summarizes some of the key factors that might be of considerations:

- End-to-end security is often hampered by the issue of how to provide secure access between middleware components that use disparate security technologies, such as user registries and security token formats.
- Often security is at odds with performance, because the most secure techniques are normally the most expensive to implement, requiring the most processing overhead.
- The range of options is vast and the required skill level is high, both of which can sometimes slow down the implementation.
- It is not always easy to establish a clear set of security requirements, thus making the job of the security architect difficult.

6.2 Traditional CICS security

In a CICS environment, the assets that you normally want to protect are the application programs and the resources that are accessed by the application programs. To prevent disclosure, destruction, or corruption of these assets, you must control access to the CICS region and to different CICS components.

You can limit the activities of a CICS user to only those functions that the user is authorized to use by implementing one or more of the following CICS security mechanisms:

- Transaction security

  This ensures that users who attempt to run a transaction are entitled to do so.
- Resource security

  This ensures that users who use CICS resources, such as files and transient data queues, are entitled to do so.
- Command security

  This ensures that users who use CICS system programming commands are entitled to do so.
- Surrogate security

  This ensures that a *surrogate* user is authorized to act on behalf of another user.

When CICS security is active, requests to attach transactions, and requests by transactions to access resources, are associated with a user ID. When a user makes such a request, CICS calls the external security manager (such as RACF) via a SAF[2] interface to determine whether the user ID has the authority to complete the request. If the user ID does not have the correct authority, CICS denies the request.

In many cases, a user is a human operator, interacting with CICS through a terminal or a workstation. However, the user can also be a web browser user or, in a web services solution, a program executing in a client system.

Identifying the user

When a human operator signs on to a CICS region at the start of a terminal session, he is challenged to provide a user ID and password. The user ID remains associated with the terminal until the terminal operator signs off. Transactions executed from the terminal, and requests made by those transactions, are associated with that user ID.

For connections from web users, there are other ways that the user of a CICS transaction can be identified, including these:

- An HTTP client can provide HTTP basic authentication information (a user ID and password). The transaction that services the client's request, and further requests made by that transaction, are associated with that user ID.
- A client program that is communicating with CICS using the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) can supply a client certificate to identify itself. The security manager maps the certificate to a user ID. The transaction that services the client's request, and further requests made by that transaction, are associated with that user ID.

In addition to these transport-level authentication mechanisms, web service clients can also pass authentication data, in the form of a security token, within the SOAP message itself. CICS provides direct support for Username tokens and X.509 certificates, and it can also interoperate with a Security Token Service (STS), such as Tivoli Federated Identity Manager, to provide more advanced authentication of web services.

For a complete discussion of traditional CICS security, refer to the *CICS TS V4.2 RACF Security Guide,* SC34-7179.

Identity assertion

Modern enterprise information processing systems typically consist of multiple software components, for example, a WebSphere Application Server running on

[2] Security Access Facility (SAF) is the high-level infrastructure that allows you to plug in a commercially available security product, such as RACF.

a distributed platform, and CICS and DB2 running on z/OS. This often introduces the challenge of how to provide secure access between middleware components that use disparate security technologies. As stated above, any user of CICS that requires access to data will usually need to provide a valid credential for authentication and subsequent authorization via RACF security manager. But what if the user does not have a RACF user ID?

The solution has often been to provide a form of *identity assertion*, where the distributed identity is mapped to a RACF identity and then asserted to CICS without a password check. However, no matter what solution is used for identity assertion, the process of mapping distributed user identities to a RACF identity has typically been a one-way function, resulting in the loss of the original distributed identity after the mapping occurs. Although effective, such mapping solutions have several issues, including lack of end-to-end accountability, inflexibility, and loss of control.

z/OS identity propagation

z/OS identity propagation is a newer form of identity assertion provided by z/OS V1R11. Together with new functions in CICS TS V4.1, and WebSphere DataPower® or CICS Transaction Gateway (CICS TG), it supports a cross-platform, end-to-end security solution, providing for identity assertion, control, and auditing.

Note: z/OS identity propagation is supported with CICS TS V4.1 with a set of enabling APARs:

- PK83741
- PK95579
- PM01622

APAR PK98426 is also required if you are using CICSPlex SM.

Identity propagation addresses the issues associated with previous identity assertion solutions by allowing the z/OS security administrator to create a set of flexible rules, stored in the RACF database, ensuring that the distributed identity persists after the mapping stage and remains visible for operational support and auditing. For information about how z/OS identity propagation can be used with CICS Web services see “z/OS identity propagation support with CICS Web services” on page 151, and for information about how z/OS identity propagation can be used with CICS TG see “z/OS identity propagation support with CICS TG” on page 163.

6.3 Cryptography

Cryptography is the scientific discipline for the study and development of *ciphers*, in particular, encryption and decryption algorithms. These cryptographic procedures are the essential components that enable secure communication to take place across networks that are not secure. SSL/TLS encryption uses both *symmetric* and *asymmetric* keys.

- Symmetric (secret) key

  Secret key cryptography means that the sender and receiver share the same (symmetric) key, which is used to encrypt and decrypt the data.

  The secret key encryption and decryption process is often used to provide privacy for high-volume data transmissions.

- Asymmetric (public/private) key

  Public/private key cryptography uses an asymmetric algorithm. The private key is known only by its owner and is never disclosed. The corresponding public key can be known by anyone. The public key is derived from the private key, but it cannot be used to deduce the private key. Either key of the pair can be used to encrypt a message, but decryption is only possible with the other key.

Most of the CICS integration options discussed in this publication support security solutions that are based on these cryptographic procedures, including SSL/TLS.

6.3.1 Transport Layer Security (TLS) 1.0 protocol

CICS supports two security protocols that can be used to provide secure communication over the internet. The first is the Secure Sockets Layer (SSL) 3.0 protocol. The second is the Transport Layer Security (TLS) 1.0 protocol, which is based on SSL 3.0.

The primary goal of SSL/TLS is to provide privacy (confidentiality) and data integrity between two applications communicating over the internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

When a TLS client and server first start communicating, a *handshake* occurs. During the handshake, the client and server agree on which version of the TLS protocol they will use, select a cipher suite, optionally authenticate each other, and use public key encryption techniques to generate shared secrets.

SSL/TLS requires a server X.509 certificate, which is stored in the server's certificate key ring. The certificate is used as part of the handshake server authentication process. The client validates the server certificate. Successful server authentication requires that the Certificate Authority (CA) that signed the server certificate be considered trusted by the client. To be considered trusted, the certificate of the CA must be in the key ring of the client.

SSL/TLS optionally uses a client X.509 certificate that is used as part of the handshake client authentication process. To use client authentication, the client must have a client X.509 certificate. The server validates the client certificate. Successful client authentication requires that the Certificate Authority (CA) that signed the client certificate be considered trusted by the server. To be considered trusted, the certificate of the CA must be in the key ring of the server.

CICS uses z/OS System SSL (a component of z/OS Communications Server) to support both the SSL and TLS protocols.

Application Transparent Transport Layer Security

Application Transparent Transport Layer Security (AT-TLS) consolidates TLS implementation in one location, such that exploiters can then use these centralized services without having to implement the TLS protocol themselves. AT-TLS is based on z/OS System SSL, and transparently implements these protocols in the TCP layer of the stack.

For TCP/IP access to CICS, you can either use the CICS-provided SSL/TLS support or AT-TLS. The CICS-provided support requires more configuration but allows client X.509 certificates to be used for authentication and identification.

See 6.11, "CICS sockets" on page 181 for an example of how AT-TLS can be used to secure TCP/IP connectivity to CICS.

6.3.2 ICSF

The Integrated Cryptographic Service Facility (ICSF) is a software element of z/OS that works with cryptographic hardware features and RACF to provide secure, high-speed cryptographic services in the z/OS environment. ICSF provides the application programming interfaces by which applications, and subsystems such as CICS, request the cryptographic services.

Note: CICS makes use of cryptographic services provided by ICSF.

6.3.3 Cryptographic hardware

When using cryptographic functions to secure CICS applications, it is important to minimize the performance overhead by utilizing hardware cryptographic devices.

The cryptographic hardware features available to your CICS regions depend on the type of System z server that you are using. These are the main cryptographic hardware capabilities available today:

- CP Assist for Cryptographic Functions (CPACF)

  CPACF offers a set of symmetric cryptographic functions available on all CPs of a zEnterprise™ server. The CPACF feature provides hardware acceleration for DES, triple-DES, AES, MAC, and SHA cryptographic services. It provides high-performance hardware encryption, decryption, and hashing support. CPACF has to be enabled to be used, but it is a non-chargeable feature (feature code 3863).

  Note: If available, CPACF will be used by System SSL for encrypting data packets, but it provides no assistance for SSL handshakes.

- Crypto Express 3 feature

  The optional Crypto Express 3 (CEX3) comes as a pluggable feature that provides a high-performance and secure cryptographic environment. Each CEX3 feature contains two cryptographic engines, each of which can be configured as an asynchronous cryptographic coprocessor (CEX3C) or accelerator (CEX3A). The CEX3A provides hardware support to accelerate the computationally intensive public key operations used by SSL/TLS during the handshake process.

6.4 z/OS Communications Server security

A range of z/OS Communications Server security functions can be used to protect access to your CICS applications. These are the two goals of z/OS Communications Server network security support are:

- To protect the mainframe from the network
- To protect critical mainframe data (such as the data accessed by CICS applications) within the network

Security functions that achieve these goals are implemented throughout the layers of the communications stack. Figure 6-1 shows the various z/OS Communications Server network security functions with these goals in mind.

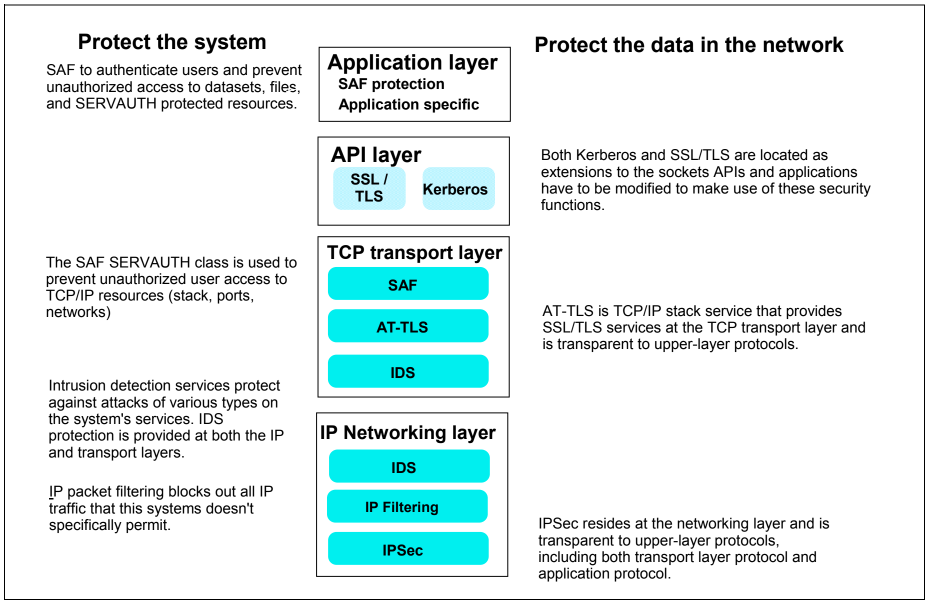

Protect the system
SAF to authenticate users and prevent unauthorized access to datasets, files, and SERVAUTH protected resources.
The SAF SERVAUTH class is used to prevent unauthorized user access to TCP/IP resources (stack, ports, networks)
Intrusion detection services protect against attacks of various types on the system's services. IDS protection is provided at both the IP and transport layers.
IP packet filtering blocks out all IP traffic that this systems doesn't specifically permit.
Application layer
SAF protection
Application specific
API layer
SSL / TLS
Kerberos
TCP transport layer
SAF
AT-TLS
IDS
IP Networking layer
IDS
IP Filtering
IPSec
Protect the data in the network
Both Kerberos and SSL/TLS are located as extensions to the sockets APIs and applications have to be modified to make use of these security functions.
AT-TLS is TCP/IP stack service that provides SSL/TLS services at the TCP transport layer and is transparent to upper-layer protocols.
IPSec resides at the networking layer and is transparent to upper-layer protocols, including both transport layer protocol and application protocol.

Figure 6-1 z/OS Communications Server security functions

First, we look at the functions that protect the z/OS system from the network, from the bottom of the network stack to the top:

- IP packet filtering is a network layer function. It blocks unwanted traffic from entering the z/OS system. In addition, it controls whether traffic is allowed to leave the system. This function is controlled with a set of filter rules, which are defined as a security policy to z/OS.
- Intrusion Detection Services are implemented at both the network and transport layers of the TCP/IP stack. It detects and protects against potentially malicious or damaging behavior directed at the system’s open network services. This function is controlled with an IDS policy that is defined to z/OS.
- SAF protection is provided at the transport layer to control local user access to TCP/IP resources such as the TCP/IP stack, network resources, and TCP and UDP ports. These TCP/IP resources are defined to SAF using the SAF SERVAUTH profile.
- Applications that are part of z/OS Communications Server use SAF for identification, authentication, and access control for files, datasets, and other application resources.

Next, we look at the security functions that protect data that flows to and from the z/OS system in the network:

- IPSec is implemented at the network layer and provides authentication, message integrity validation, and encryption for network traffic. IPSec can be configured at a very wide scope to cover all traffic between two systems, or at a narrow scope to protect traffic for specific applications. Because it is implemented at the IP layer, it can protect all application protocols without requiring any application changes.
- AT-TLS applies SSL or TLS protection to inbound and outbound network traffic within the transport layer.

  Both IPSec and AT-TLS are controlled using a network security policy.
- z/OS provides SSL/TLS application interfaces so that applications can continue to access SSL/TLS services as a sockets layer service.

  Kerberos services are also available for applications as a sockets layer service.

6.5 Technology comparison table

This section describes which security models are supported by the various access technologies that CICS supports. Table 6-1 provides this comparison in table form. Refer to the following sections for a more detailed explanation.

Table 6-1 Security models: Technology comparison table

	CICS Web services	**CICS TG for z/OS**	**WOLA**	**CICS web support**	**WebSphere MQ**	**CICS sockets**
Basic authentica-tion	Supported	Supported	Not supported	Supported	Supported	Supported
Identity assertion	Supported	Supported	Supported	Supported	Supported	Supported
z/OS identity propagation	Supported when using DataPower and CICS TS V4.1	Supported when using JCA with CICS TG V8 and CICS TS V4.1	Not supported	Not supported	Not supported	Not supported
SSL/TLS	Supported	Supported	Supported	Supported	Supported	Supported with AT-TLS

	CICS Web services	CICS TG for z/OS	WOLA	CICS web support	WebSphere MQ	CICS sockets
Message security	Supported with WS.* standards	Not supported	Not supported	Not supported	Supported with WMQ AMS	Not supported

6.6 CICS Web services

Whereas service enablement improves openess and business agility, it also introduces new security risks. These security risks need to be countered by measures that protect access to the service and ensure confidentiality of data as it is transported across the SOA infrastructure. CICS supports a wide range of transport-based and message-based security options.

Note: A security solution can use a combination of transport-based and message-based security mechanisms.

6.6.1 Transport security

Transport-based security is normally the first option considered for CICS Web services. Different transport security options are available depending on the transport used for invoking a CICS Web service.

Transport security options for HTTP

When a CICS Web service is invoked using HTTP, you can use basic authentication to authenticate the web service client, and you can also use SSL/TLS to both authenticate the client and to ensure message integrity and confidentiality:

- CICS service provider and requester applications can be protected by HTTP basic authentication (see “CICS web support security” on page 172).
- CICS service provider and requester applications can be secured using HTTPS (HTTP over a SSL/TLS connection). See “CICS web support security” on page 172 for more information about using HTTPS with CICS.

Transport security options for WebSphere MQ

When a CICS Web service is invoked using WebSphere MQ (WMQ), you can use any of the WMQ built-in security mechanisms or you can use message security, or a combination of both. See “Security considerations for WebSphere MQ” on page 180 for more information about WMQ security options.

6.6.2 SOAP message security

Transport-based security mechanisms are mature and have been optimized over a long period of time. However, basic authentication and SSL/TLS are *point-to-point* security mechanisms and might not be suitable for more complex configurations that involve intermediaries. As illustrated in Figure 6-2, if the service requester identifies itself to the intermediate server (for example, WebSphere Application Server or an ESB), and the intermediate server identifies itself to the service provider, the target service will normally run with the identity of the intermediate gateway rather than the service requester.

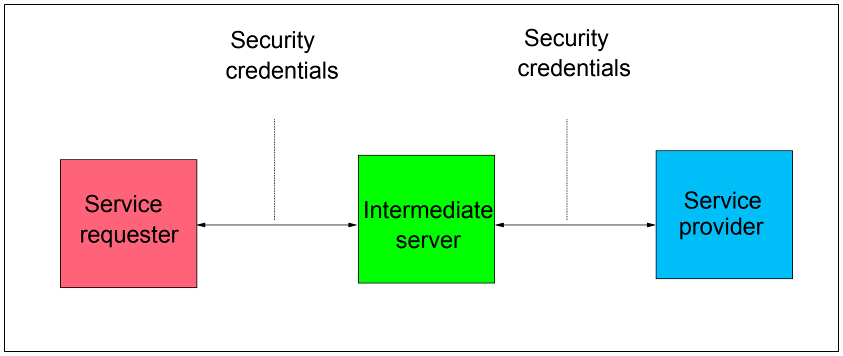


Figure 6-2 Transport-level security with an intermediate server

The *WS-Security* specification provides a foundational set of SOAP message extensions for building secure web services by defining new elements to be used in the SOAP header for message-level security. It specifies the use of *security tokens, digital signatures*, and *XML encryption* to protect and authenticate SOAP messages. It specifies the use of digital signatures to provide integrity for XML elements in a SOAP message, and it specifies the use of encryption to provide confidentiality for XML elements in a SOAP message. The specification allows you to protect the body of the message or any XML elements within the body or the header. You can give different levels of protection to different elements within the SOAP message.

One advantage of using WS-Security over SSL/TLS is that it can provide *end-to-end* message-level security. This means that the service can run with the identity of the service requester because the requester's identity can flow with the message across the intermediary servers. Message security can also be protected even if the message goes through multiple *untrusted* intermediaries (Figure 6-3).

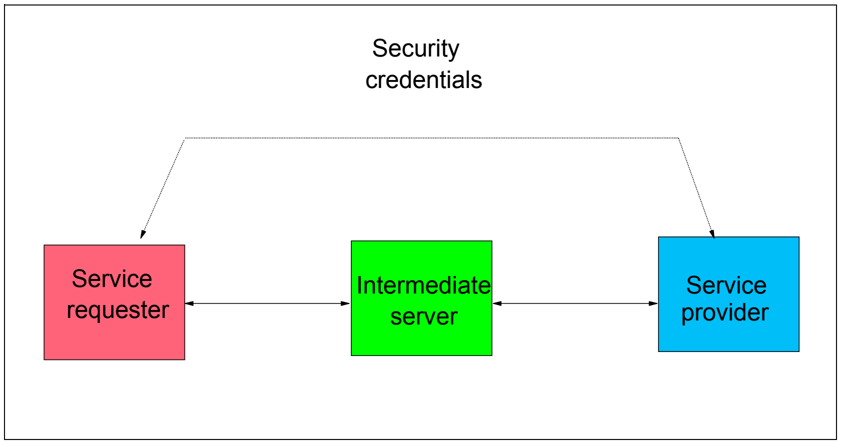


Figure 6-3 SOAP message security with an intermediate server

Figure 6-4 shows how a SOAP message can be extended with security data that is used to authenticate the service requester and to protect the message as it passes between the service requester and the CICS service provider. The network portion of the diagram could contain any number of intermediate nodes, some of which might not be trusted.

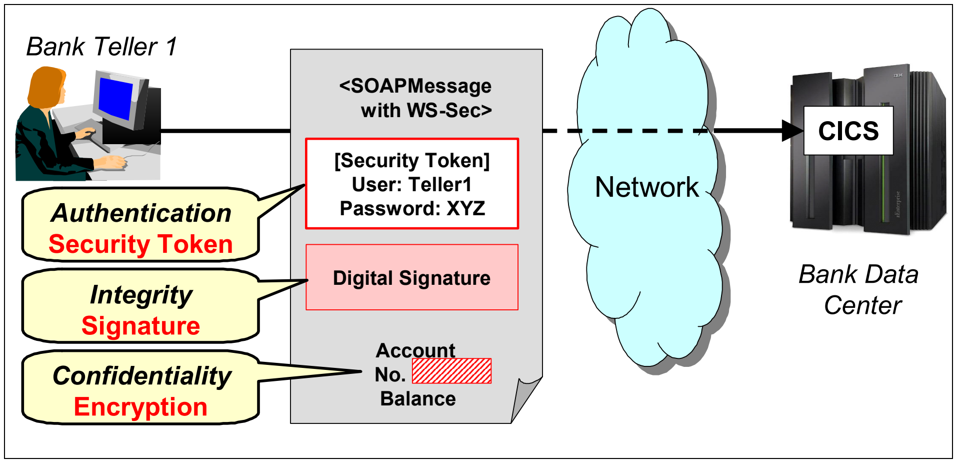

Bank Teller 1
<SOAPMessage with WS-Sec>
[Security Token] User: Teller1 Password: XYZ
Digital Signature
Account No. Balance
Authentication Security Token
Integrity Signature
Confidentiality Encryption
Network
CICS
Bank Data Center

Figure 6-4 An example of a typical scenario with WS-Security

The SOAP message shown in Figure 6-4 on page 146 contains three pieces of security data:

- A security token used to authenticate and identify user Teller1
- An XML digital signature to ensure that no one modifies the message while it is in transit without the modification being detected
- An account balance XML element that is encrypted to ensure confidentiality

CICS also supports the *WS-Trust* specification, which provides a framework for requesting and issuing security tokens and managing trust relationships between web service requesters and providers. This extension to the authentication of SOAP messages enables CICS Web services to validate and exchange security tokens of different types using a trusted third party known as a Security Token Service (STS). IBM Tivoli Federated Identity Manager (TFIM) can act as an STS by providing the necessary framework to support standards-based, federated identity management between enterprises that have established a trust relationship.

To read more about the WS-Security and WS-Trust specifications, refer to this website:

`http://www.oasis-open.org/`

CICS supports a wide range of message-based options for securing SOAP messages:

- Basic authentication

  In service provider mode, CICS can accept a Username token in the SOAP message header for authentication on inbound SOAP messages. The Username token contains a username element and a password element. CICS verifies the username and password using an external security manager such as RACF. If this is successful, CICS sets the user ID of the pipeline task to this value.

  Username tokens that contain a password are not supported on outbound SOAP messages when CICS is the service requester. This is because CICS does not have access to a user’s password. However, if there is a requirement to do this, then it is possible to write a header processing program that adds a Username token to the outbound SOAP message.

- Advanced authentication

  In service provider and requester pipelines, you can verify or exchange security tokens with a Security Token Service (STS) for authentication purposes. The STS enables CICS to accept and send messages that have security tokens in the message header that are not normally supported, for example, LTPA and Kerberos tokens or SAML assertions.

For an inbound message, you can select to verify or exchange a security token. If the request is to exchange the security token, CICS must receive a Username token back from the STS. For an outbound message, you can only exchange a Username token for a security token.

- Signing with an X.509 certificate

  In service provider and service requester mode, you can digitally sign a part of the SOAP message to ensure that the data is not modified in transit.

  An X.509 certificate that is used for signing can also be used for authentication. This type of security token is known as a *binary security token*. In service provider mode, CICS maps the certificate to a RACF user ID and sets the user ID of the pipeline task to this value. In service requester mode, CICS can send an X.509 certificate in the SOAP message header to the service provider that is then used for authentication purposes.

  Important: ICSF must be started and configured with cryptographic devices in order to use the CICS WS-Security XML digital signature and XML encryption support.

- Encrypting

  In service provider and service requester mode, you can use XML encryption to encrypt the SOAP message body using a symmetric algorithm such as Triple DES or AES. A symmetric algorithm is where the same key is used to encrypt and decrypt the data.

- Identity assertion

  In service provider pipelines, CICS can accept a Username token or X.509 certificate in the SOAP message header as an asserted identity. When a Username token is used, the password is not required. When an X.509 certificate is used the certificate is mapped to a RACF user ID. CICS trusts the provided identity and sets the user ID of the pipeline task to this value.

  In service requester pipelines, CICS can send a Username token without the password in the SOAP message header to the service provider.

- z/OS identity propagation

  In service provider mode, you can use an unauthenticated Extended Identity Context Reference (ICRX). An ICRX identity token is a z/OS identifier that maps to a user ID. CICS resolves the ICRX identity token to a RACF user ID and sets the user ID of the pipeline task to this value. For more information about z/OS identity propagation see “z/OS identity propagation support with CICS Web services” on page 151.

6.6.3 Java-based SOAP pipeline

In CICS TS V4.2, as well as the native SOAP pipeline, CICS supports using the Axis2 Java-based SOAP engine to process web service requests in provider and requester pipelines. When using a Java-based SOAP pipeline, you should be aware of the following additional security considerations:

- All transport-based security mechanisms (for example, SSL/TLS) are supported.
- For non-Java web service applications, where the Java-based SOAP pipeline is being used in place of the normal SOAP pipeline, all SOAP message-based security mechanisms continue to be supported.

  For Axis2 web service Java applications, for example, JAX-WS applications, where the complete processing of the web service invocation takes place within a JVM, SOAP message-based authentication and identification (including identity assertion and z/OS identity propagation) are not supported.

Restriction: Authentication and identification using SOAP message security are not supported for Axis2 web service Java applications.

6.6.4 Using an SOA appliance to secure CICS Web services

An SOA appliance can be used as an SOA Gateway or an ESB. One role of the appliance can be to secure service requests, thus offloading expensive XML and cryptography processing from the target server that runs the service.

The WebSphere DataPower XI52 is an SOA appliance that is well known for its security features and its high throughput in XML processing. It can be integrated in a DMZ and it can detect and reject XML attacks. The appliance can act as an XML accelerator and transformation engine, be used as a firewall and security device (authentication, authorization, auditing, encryption and decryption, and so on), and also function as an Enterprise Service Bus (ESB). It can perform complex security checks without performance degradation.

DataPower can be used in conjunction with CICS Web services to help secure the services and to offload expensive operations by processing the complex part of XML messages (such as an XML digital signature) at wirespeed (Figure 6-5).

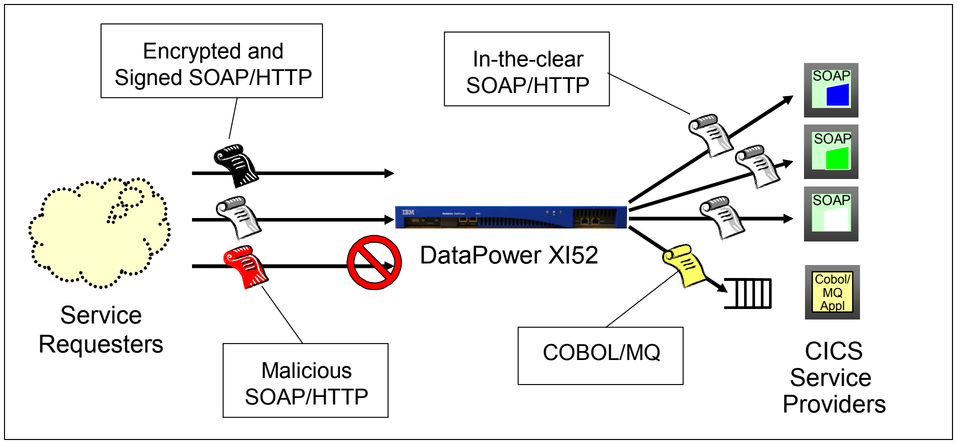


Figure 6-5 Using a WebSphere DataPower SOA Appliance with CICS Web services

DataPower is typically used in the following scenarios:

- To process encrypted or signed SOAP or XML messages, thus off-loading some of the CPU-intensive XML processing from CICS
- To intercept and reject malicious SOAP or XML messages
- To transform XML data to non-XML data, for example, COBOL binary data
- To switch from the HTTPS protocol to another protocol, for example, HTTP or WebSphere MQ, thus off-loading the SSL/TLS processing from CICS

These DataPower capabilities are also available in the DataPower XI50z, which is a blade-form factor that is installed in the zEnterprise Blade Extension (zBX). Some of the additional benefits from using DataPower XI50z include these:

- Secure integration between DataPower and the virtual servers within the zEnterprise through the use of the high-speed intraensemble data network (IEDN)
- Extended ESB integration across the zEnterprise
- Centralized installation, operations, and management of DataPower using the Unified Resource Manager[3]

[3] The Unified Resource Manager provides integrated management across all elements of the zEnterprise.

To read more about DataPower XI52 and DataPower XI50z, refer to this URL:

http://www.ibm.com/software/integration/datapower/xi50/#

z/OS identity propagation support with CICS Web services

WebSphere DataPower can be used to implement identity propagation with CICS Web services. Figure 6-6 shows an overview of a CICS Web services identity propagation scenario.

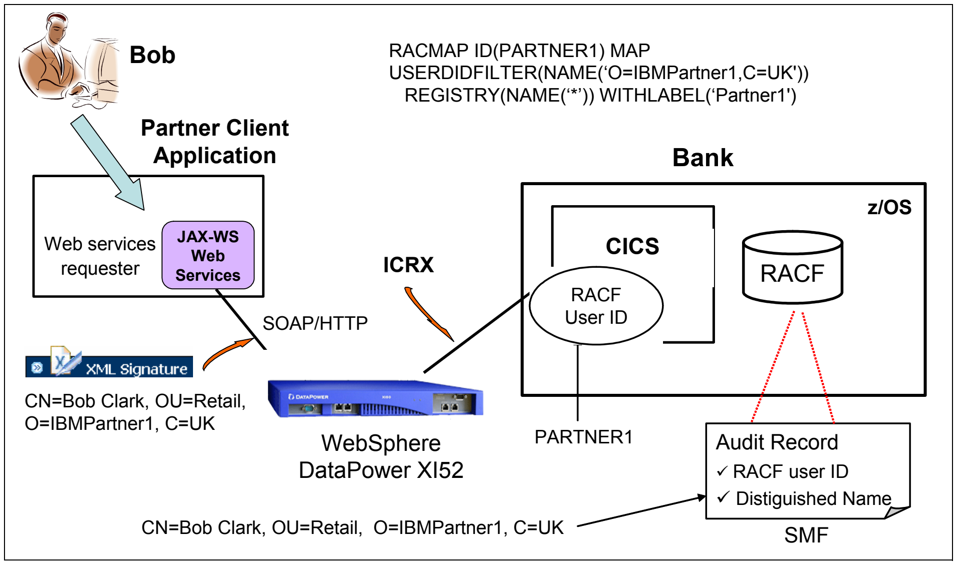


Figure 6-6 z/OS identity propagation with CICS Web services

Figure 6-6 shows an employee of the business partner of a bank (Bob) who makes a web service call that is authenticated by WebSphere DataPower. After successful authentication, WebSphere DataPower propagates the user's distributed identity (in the form of a distinguished name) to the CICS core banking application. The bank has a requirement to authorize requests based on a generic RACF user ID that represents the business partner, but also to keep an audit trail of which business partner employee invoked the service.

This is the sequence of processing steps:

1. A partner employee Bob uses a partner client application that sends a web service request to the bank.
2. The service requester application generates a BinarySecurityToken element from Bob's X.509 certificate, signs the message with Bob's private key, and sends the request to the target endpoint, which is configured to be the DataPower appliance.

3. DataPower verifies the XML digital signature.

> **Note:** DataPower provides a range of authentication mechanisms including LDAP authentication, SSL client authentication, SAML authentication, and many more.

4. DataPower extracts the identity of the service requester (**`CN=Bob Clark, OU=Retail,O=IBMPartner1,C=UK`**) using the certificate passed as part of the <X509/> element of the digitally signed message.
5. DataPower authenticates the user by validating the signer certificate of the digitally signed message.
6. DataPower propagates Bob's identity (Bob's DN) to CICS in the form of an ICRX over a trusted SSL connection.
7. CICS receives the SOAP message from DataPower. The PIPELINE configuration file includes the CICS-supplied WS-Security handler program which locates the ICRX in the WS-Security header and uses the ICRX to identify the user.
8. CICS issues a RACROUTE REQUEST=VERIFY to map the ICRX into the RACF user ID PARTNER1.

> **Important:** This mapping is based on the **`RACMAP`** command issued by the security administrator that maps all employees of the business partner to the same RACF user ID.

9. The CICS task runs under the mapped RACF user ID (PARTNER1) but retains the association with the original distributed identity (CN=Bob Clark, OU=Retail, O=IBMPartner1, C=UK).

The advantage of this solution is that the original caller's identity is not lost. It is stored as an extension to the RACF identity. An interesting aspect of ICRX-based identity propagation is that it makes shared role-based RACF user IDs (DEV/MANAGER/SYSADMIN/OPERATOR, and so on) more desirable than in the past because an audit trail to a specific individual can be maintained even if the target RACF identity is shared.

Trust

For identity assertion with web services, the intermediate server should establish a *trust relationship* with the CICS region by authenticating itself and then by being recognized as a trusted partner of the CICS region. CICS supports two different models for establishing this trust relationship:

- Trust token

  The intermediary server sends a trust token to CICS.

- Blind trust

  Trust is established at the transport level, for example, with SSL client authentication.

The CICS Web services identity propagation support is normally used with the blind trust model, which has the advantage that the trust established between the intermediary server and CICS can be persistent. This can occur, for example, by using SSL persistent connections or a Virtual Private Network (VPN). It does not need to be re-established for each SOAP message. When using SSL client authentication to establish the trust relationship, the SSL certificate that WebSphere DataPower uses to identify itself can be associated with a RACF user ID, and *surrogate user checking* can then be used to authorize this user ID to assert the RACF ID that is mapped from the distributed identity.

Note: A *surrogate* user is a RACF user ID that is authorized to act on behalf of another user (the original user).

For detailed information about configuring identity propagation with CICS Web services and WebSphere DataPower refer to *z/OS Identity Propagation*, SG24-7850.

6.6.5 Security considerations for CICS Web services

Figure 6-7 shows a typical web services security scenario in which a client makes a call to an intermediate server, which then makes a web service call to CICS.

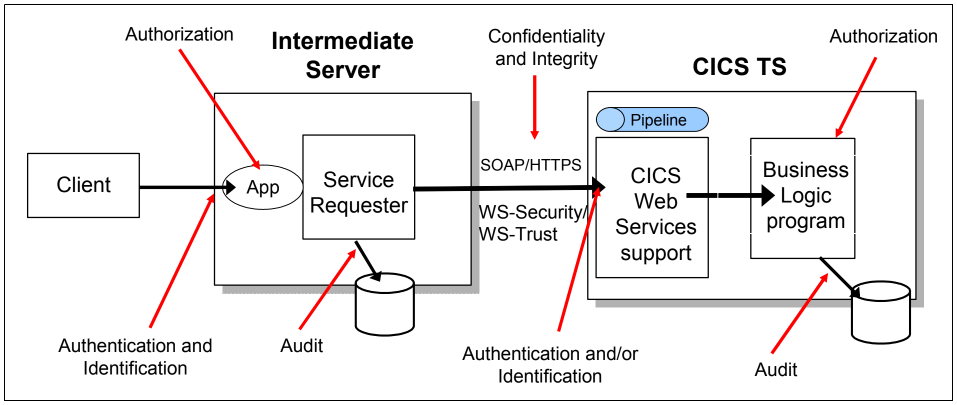


Figure 6-7 CICS Web services security questions

The following sections containing lists of questions and comments will help you to choose between the different options available for securing the CICS Web service shown in Figure 6-7.

Authentication

Consider the following authentication-related questions:

- Does the service requester need to authenticate?

  This can be decided for specific services rather than there being a general rule for the application. It might be appropriate to run read-only services using a generic user ID, whereas more sensitive services might need the requester to authenticate. In CICS, this split can be made by running secured services on a different pipeline to unsecured services.

- Who authenticates the service requester? CICS or an intermediary server?

  CICS can authenticate service requesters directly, or an intermediary might be able to provide an authentication service to CICS. In this case, the intermediary server authenticates the service requester and then flows an *asserted* identity to CICS.

- Will you use transport-based or SOAP message based authentication?

  You might choose to use only transport-based security to secure your CICS Web services environment in these circumstances:

  – No intermediaries are used in the web service environment. Or, if there are intermediaries, you can guarantee that after the data is decrypted, it cannot be accessed by an untrusted node or process.
  – The transport is only based on HTTP.
  – Performance is your primary concern (see “Performance” on page 158).
  – The web services client is a stand-alone Java program.

    WS-Security can only be applied to clients that run in a web services environment that supports the WS-Security specification (for example, WebSphere Application Server).

  You might choose to use WS-Security (possibly in addition to transport-level security) in these circumstances:

  – Intermediaries are used, some of which might be untrusted.

    Security credentials that flow in the SOAP message can pass through any number of intermediaries. Protecting confidential information in the actual SOAP message can avoid the overhead of encrypting and decrypting via SSL at every intermediary node.
  – Multiple transport protocols are used.

    WS-Security works across multiple transports and is independent of the underlying transport protocol.
  – You might choose to implement your own security procedures and processing by writing a custom message handler program that can process secure SOAP messages in the pipeline.

- If you chose SOAP message-based security, what token type will be used?

  – Username tokens, X.509 certificates, and ICRX tokens can be processed directly by the CICS-supplied security handler.
  – You will most likely have to configure the CICS-supplied handler to call an STS if other token types are used.
  – You can also use an STS to process non-standard token types or you can write a custom security handler.

Identification

Consider the following identification-related question: How will you assign the RACF user ID for running the CICS business logic program?

- It might come from the authentication credentials.
- It might be asserted by an intermediate server, either as a Username token, X.509 certificate, or a ICRX.
- It might be assigned by a user-written message handler that runs as part of the pipeline processing.
- It might be hard-coded in CICS.

Authorization

Consider the following authorization-related question. Does the CICS task need to run with the service requester's identity?

If it is not necessary to run the CICS task with the service requester's identity, a RACF user ID can be specified in a URIMAP (this avoids running the web service with the CICS default user ID).

If it is necessary to run the CICS task with the service requester's identity:

- In the case where no intermediaries are used, CICS authorization processing can be based on the RACF user ID that is associated with the security token that is used by the requester to authenticate.
- In the case where intermediaries are used, CICS authorization processing can be based on the asserted identity token that is passed by the intermediary.

Surrogate authorization checking should be used to ensure that the intermediary has the correct authority to start work on behalf of the asserted identity.

Integrity

Consider the following integrity-related questions:

- Does the integrity of the data warrant protection?

  If SOAP messages do not contain critical data, or if the messages are only transmitted within an internal secure network, then it might be reasonable to flow unsigned messages.

  A single CICS region can process signed and unsigned messages. For example, you can define pipelines that expect to receive signed messages and other pipelines that will reject signed messages.

- Are intermediaries used?

  SSL/TLS only provides integrity of the data during the message transmission. XML signatures might be necessary to protect message integrity within every intermediary node.

- Does CICS need to deal with signed messages?

  It might be appropriate for an intermediary server to terminate an HTTPS session and forward the request to CICS across a secured network as an HTTP request.

  Equally, it might be appropriate for an intermediary server to validate an XML digital signature and forward an unsigned message to CICS. In this case, it is still possible for the service requester's identity to flow with the unsigned message so that it can be used for CICS resource authorization checking.

Confidentiality

Consider the following confidentiality-related questions:

- Does the sensitivity of the data warrant encryption?

  If SOAP messages do not contain sensitive data, or if the messages are only transmitted within an internal secure network, then it might be reasonable to flow unencrypted messages.

- Are intermediaries used?

  SSL/TLS only provides privacy of the data during the message transmission. Protecting confidential information in the actual SOAP message using XML encryption might be necessary to protect message confidentiality within every intermediary node.

  Be aware that XML encryption might make it difficult to perform content-based routing of SOAP messages because the intermediary server will not be able to read all parts of the message body.

- Does CICS need to deal with encrypted messages?

  It might be appropriate for an intermediary server to terminate an HTTPS session or to decrypt XML-encrypted messages and forward unencrypted messages to CICS.
- Does CICS call an STS?

  You should use SSL/TLS to keep the connection to the STS secure.

Auditing

Consider the following auditing-related questions:

- Does the service request need to be audited?

  Typical information that needs to be captured is the RACF user ID used for running the CICS task, the operation, and the nature of the data update.
- Do you need an audit log that contains the original distributed identity?

  Consider the z/OS identity propagation support that provides for identity assertion, control, and auditing as part of the built-in processing within CICS, RACF, and DataPower.
- Will you create an audit trail in CICS or in an intermediary server, or both?

  It might be that you need an audit trail in all the servers that are involved in processing a request.

  DataPower is an ideal place to perform auditing because it provides a configurable solution that supports multiple format log records that can be stored on the appliance or transferred off-device.

Performance

Security is often at odds with performance. That is, the most secure technologies are often expensive to implement. Inevitably, the need to implement a solution that meets the security requirements must be balanced against the need to meet the solution’s performance objectives.

After you have a clear understanding of the security requirements and security implementation options, consider the following CICS performance-related questions.

- Will you use transport-based security or SOAP message security?

  SSL/TLS is a mature technology that has been optimized over a long period of time, and there are ways of optimizing performance such as persistent TCP/IP connections and SSL session ID reuse. These optimizations mean that expensive security functions, such as SSL handshaking, can be avoided for service requests following the initial handshake.

  WS-Security support, in comparison, is completely stateless, and expensive security functions, such as XML digital signature validation, are repeated for each service request.

  In practice, the most optimum solution is often to use a combination of transport-based and SOAP message based security, for example, transport-based security for confidentiality and data integrity and SOAP message-based security for transport of security tokens.

- Where is authentication done and how many times?

  The cost of authentication in CICS is dependent on the security token used in the authentication. Simple security tokens like UsernameTokens are less expensive than binary security tokens such as X.509 certificates. More advanced authentication using a STS will incur an additional overhead.

  If authentication is done by an intermediary server, where possible, the intermediary server should flow an asserted identity to CICS. This avoids the overhead of authenticating multiple times.

- Are you using hardware cryptographic devices?

  Cryptographic hardware and Integrated Cryptographic Hardware Facility (ICSF) are prerequisites for CICS XML digital signature and XML encryption processing. They are also required in order to maximize performance when CICS is configured to use SSL/TLS.

- What cipher suite are you using?

  To utilize hardware cryptography, the chosen cipher suite algorithm must be available in hardware.

- Is there a need for an SOA appliance?

  An SOA appliance such as WebSphere DataPower can be used in conjunction with CICS Web services to help secure the services and to offload expensive operations by processing the complex part of XML messages (such as an XML signature) at wirespeed.

6.7 CICS TG for z/OS

In this section, we look into the security issues that arise when using the CICS TG to connect to CICS. We consider the security options when using the JCA resource adapter and also when connecting from an ECI Version 2 client.

6.7.1 JCA and security

The JCA defines a standard set of system-level contracts between a JEE application server and a resource adapter. These system-level contracts define the scope of the managed environment that the JEE application server provides for JCA components. One of the standard contracts is the security management contract that enables secure access to an EIS. Both container-managed sign-on (in which the JEE application server is responsible for flowing security context to the EIS) and component-managed sign-on (in which the application is responsible for flowing security context to the EIS) are supported.

When deploying a JEE component, the application deployer must set the *res-auth* element in the deployment descriptor to indicate which method is being used:

- Container managed security

  If you are using container-managed security, you must set the res-auth deployment descriptor element to `Container`. The application deployer must set up the authentication information (for example, set the user ID and password to be used for the connection). In some circumstances, the container can derive an identity to use from the currently executing Java principal. The application uses the `getConnection()` method of the connection factory and lets the application server manage the security to sign on to CICS.

- Component-managed security

  For component-managed security, the res-auth element needs to be set to `Application`. The application code can then supply the user ID and password when making the connection.

Different applications can use different security methods (container-managed or component-managed) based on the application deployment descriptor. However, container-managed security is normally recommended because it is a good practice to separate the business logic of an application from qualities of service such as security.

6.7.2 CICS TG for z/OS security

When using the JCA with CICS TG for z/OS, WebSphere Application Server normally accesses CICS through a Gateway daemon running on z/OS (Figure 6-8).

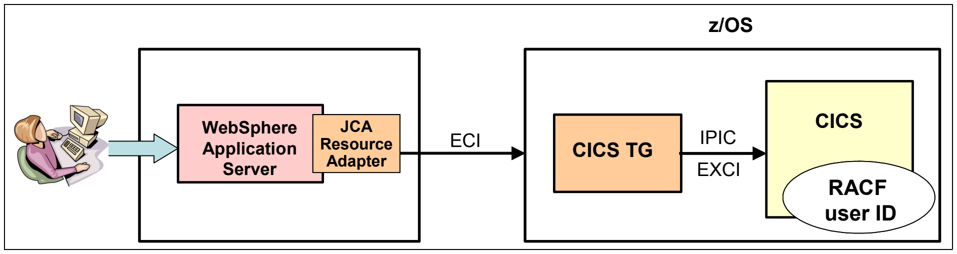


Figure 6-8 CICS TG for z/OS

In Figure 6-8, the JCA resource adapter provided by CICS TG is used to send an External Call Interface (ECI) request to CICS. The Gateway daemon is the entry point to the System z platform in which the CICS system is running, so it is normal to secure incoming ECI requests from clients.

The CICS TG for z/OS supports the following options for securing ECI requests:

- Basic authentication

  When using an EXCI connection between the Gateway daemon and CICS, the Gateway daemon can be configured to validate a user ID and password for each ECI request.

  When using an IPIC connection between the Gateway daemon and CICS, CICS can be configured to validate a user ID and password for each ECI request.

  Note: A *passphrase* can also be used with CICS TG V8.1. A passphrase is similar to a password in usage, but is generally longer for added security.

- Identity assertion

  After the user has authenticated to WebSphere Application Server, the user's password is unlikely to be available to send to the Gateway daemon. In this case, the Gateway daemon and CICS can be configured to accept a user ID without a password. In this case, establish a trust relationship between the application server, the Gateway daemon, and the CICS server (see "Trust" on page 162).

- Identity propagation

  This is a unified security solution that provides additional accountability, which is achieved by passing a distributed identity to CICS instead of a user ID and password.

  This is another form of identity assertion, and therefore you should establish the required trust relationships (see “Trust” on page 162).

- SSL/TLS

  SSL/TLS can be used for confidentiality, data integrity, and optionally for X.509 certificate authentication.

Trust

For identity assertion and identity propagation with CICS TG, we recommend that the connection between WebSphere Application Server and the Gateway daemon be configured as a trusted connection using one of the following ways:

- Using SSL client authentication to authenticate the application server to the Gateway daemon.
- Using a Virtual Private Network (VPN) or other network security configuration (see “z/OS Communications Server security” on page 141).
- Using a CICS TG security exit that allows simple pre-configured rules to be set, ensuring that only specific application servers with a known key can connect. For more information about using a CICS TG security exit, refer to the CE51 SupportPac available here:

  `ftp://ftp.software.ibm.com/software/htp/cics/support/supportpacs/individual/ce51.pdf`

A trust relationship should also be configured between the Gateway daemon and the CICS server in one of the following ways:

- When an EXCI connection is used, trust can be established using the multiregion operation (MRO) *bind* and *link* security mechanisms:

Bind security	Verifies that the system wanting to connect (bind) to CICS is authorized to do so.
Link security	The link user ID is the user ID associated with the Gateway daemon, which is passing the request to CICS. The link user ID, like the user ID flowed with the message, must be authorized to access all transactions and resources invoked as a result of the request.

  Surrogate security checks can also be enabled to confirm that the user ID associated with the Gateway daemon (the link user ID) has the appropriate authority to flow a specific user ID to CICS.

- When an IPIC connection is used, trust should be established in one of the following ways:
  - Using a pre-defined CICS IPCONN resource definition, which is a basic form of security that only allows a Gateway daemon configured with the correct name to connect to CICS
  - Using sysplex sockets, which can ensure that if non-SSL IPIC connections are used into CICS then they must come from an IP stack on the same sysplex
  - Using NET ACCESS zones, which are RACF-based rules used by the IP stack that can prevent/allow preselected IP addresses from connecting to a given TCP/IP service in use by CICS
  - Using a firewall technology

z/OS identity propagation support with CICS TG

Identity propagation provides a new security architecture for controlling identity assertion when connecting JEE applications to CICS. Identity propagation provides for a variety of identity assertion configurations when the WebSphere Application Server and CICS components are connected in a secure topology. When using the Gateway daemon on z/OS, identity propagation is only supported when the Gateway daemon and CICS region are located on the same sysplex.

Figure 6-9 shows an overview of a CICS TG identity propagation scenario.

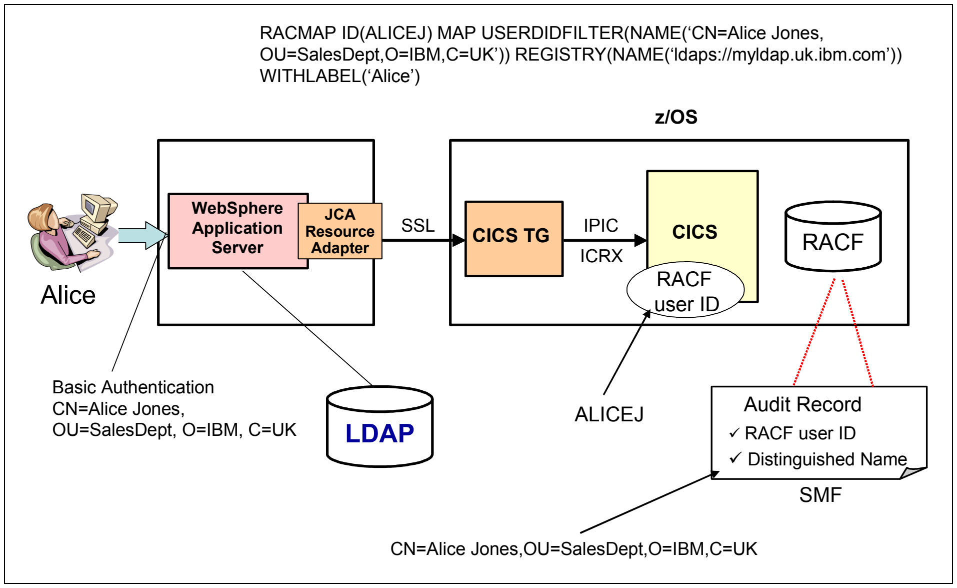

RACMAP ID(ALICEJ) MAP USERDIDFILTER(NAME('CN=Alice Jones, OU=SalesDept,O=IBM,C=UK')) REGISTRY(NAME('ldaps://myldap.uk.ibm.com')) WITHLABEL('Alice')
z/OS
Alice
WebSphere Application Server
JCA Resource Adapter
SSL
CICS TG
IPIC
ICRX
CICS
RACF user ID
RACF
Basic Authentication CN=Alice Jones, OU=SalesDept, O=IBM, C=UK
LDAP
ALICEJ
Audit Record
✓ RACF user ID
✓ Distinguished Name
SMF
CN=Alice Jones,OU=SalesDept,O=IBM,C=UK

Figure 6-9 z/OS identity propagation with CICS TG

Note: Identity propagation is supported only when using ECI calls over IPIC connections to CICS.

Figure 6-9 on page 163 shows an employee (Alice) who logs on to the company's WebSphere Application Server. After successful authentication, the WebSphere application makes a JCA call to CICS. The company has a requirement to authorize requests based on Alice's RACF user ID.

The sequence of processing steps is as follows:

1. Alice logs on to the application server and authenticates with her distributed identity (`CN=Alice Jones,OU=SalesDept,O=IBM,C=UK`).
2. The CICS TG provided login module that the JEE application is configured to use attaches Alice's distributed identity in the form of Distinguished Name (DN), onto the outbound request to the CICS TG.
3. The SSL connection from the application server to the Gateway daemon is client authenticated, thus establishing a trust between the two servers.
4. The Gateway daemon receives Alice's DN and flows it as part of an Extended Identity Context Reference (ICRX) to CICS.
5. CICS receives the request from the Gateway daemon and uses the ICRX to identify the user.
6. CICS issues a request to RACF (RACROUTE REQUEST=VERIFY) to map the ICRX into the RACF user ID ALICEJ.

   Important: This mapping is based on the `RACMAP` command issued by the security administrator that maps Alice's distributed identity to her RACF user ID.

7. The CICS task runs under the mapped RACF user ID (ALICEJ) but retains the association with the original distributed identity (CN=Alice Jones,OU=SalesDept,O=IBM,C=UK).

Important: The advantage of this solution is that the original caller's identity is not lost. It is stored as an extension to the RACF identity.

SSL/TLS support with CICS TG

The CICS TG for z/OS provides SSL/TLS support via the Java Secure Sockets Extension (JSSE). SSL/TLS provides integrity and confidentiality for the data passed between the client and the Gateway daemon, and, optionally, for authentication using a client X.509 certificate.

The features of the SSL/TLS support available with CICS TG z/OS include these:

- RACF keyring support

  SSL keystores can be stored in a RACF database.
- System z hardware cryptographic support

  This provides the ability for the CPU to offload SSL handshakes to hardware. This can substantially reduce the CPU cost of SSL handshakes and SSL data encryption.
- SSL cypher suite selection

  The choice of SSL cypher suite can be configured.

6.7.3 ECI Version 2 and security

When connecting from an ECI Version 2 client, the most likely security implementation will be basic authentication because SSL/TLS is not supported with ECI Version 2 clients.

Restriction: The ECI Version 2 client can only communicate with a CICS TG using the TCP network protocol. The use of SSL/TLS is not supported.

The Gateway daemon and CICS can be configured to accept an asserted user ID without a password. However, some form of network security will be required (see 6.4, “z/OS Communications Server security” on page 141) to make sure that the request is coming from a trusted client.

Note: In CICS TG V8.1, the ESI is supported with ECI Version 2 clients.

6.7.4 External Security Interface (ESI)

The ESI is used for verifying and changing the user ID and password information held in the CICS external security manager (ESM), such as RACF.

ESI calls to CICS can be made from Java, .NET, or C clients. It provides two basic functions:

Verify password	This allows a client application to verify a password for a given user ID.
Change password	This allows a client application to change the password for a given user ID.

Note: Prior to CICS TG V8.1, only the CICS TG Multiplatforms products supported ESI and only with SNA APPC connections. CICS TG V8.1 extends the ESI support to CICS TG for z/OS, and the ESI APIs are also enhanced to support password phrases.

6.7.5 Security considerations for CICS TG

Figure 6-10 shows two CICS TG security scenarios:

- Any client makes a call to a WebSphere Application Server, which then makes a JCA call to CICS using the CICS TG resource adapter.
- An ECI Version 2 client makes a call to CICS via the CICS TG.

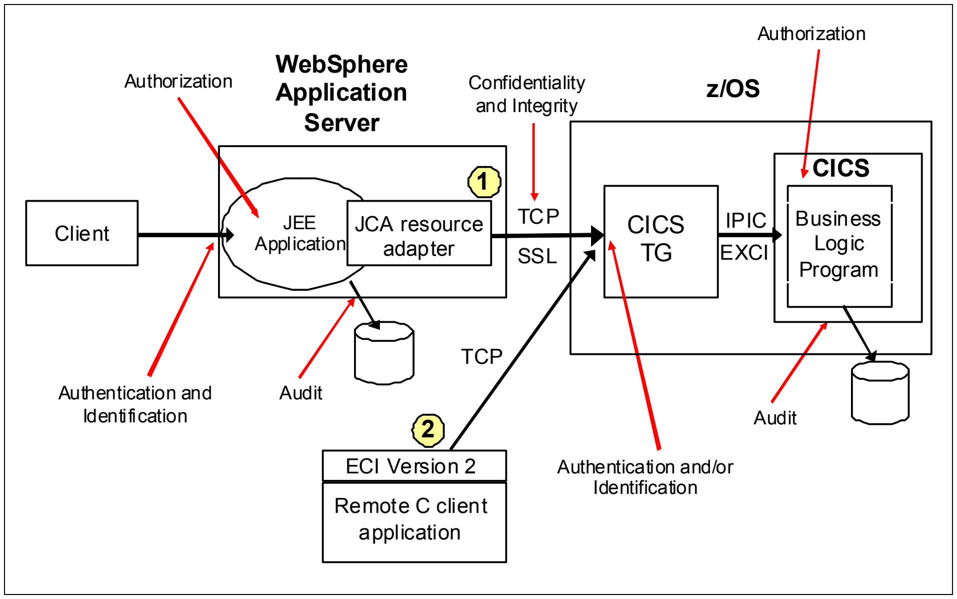

Authorization
WebSphere Application Server
Confidentiality and Integrity
Authorization
z/OS
1
Client
JEE Application
JCA resource adapter
TCP
SSL
CICS TG
IPIC
EXCI
CICS
Business Logic Program
TCP
Authentication and Identification
Audit
2
ECI Version 2
Remote C client application
Authentication and/or Identification
Audit

Figure 6-10 CICS TG security questions

The following list of questions and comments will help you to choose between the different options available for securing the ECI requests to CICS shown in Figure 6-10.

Authentication

Consider the following authentication-related questions:

- Does the requester need to authenticate?

  This can be decided for specific requests rather than a general rule for all requests. It might be appropriate to run one Gateway daemon for unauthenticated requests and another for authenticated requests.

- Who authenticates the requester, CICS TG or an intermediary server?

  CICS TG can authenticate requesters directly, or an intermediary might be able to *assert* an identity to CICS.

Identification

Consider the following identification-related question. How will you assign the RACF user ID for running the CICS business logic program?

- It might come from the authentication credentials presented to the Gateway daemon.
- It might be asserted by WebSphere Application Server, either as a RACF user ID or an ICRX.
- It might be hard-coded in the CICS.

Authorization

Consider the following authorization-related question. Does the CICS task need to run with the requester's identity?

If it is not necessary to run the CICS task with the requester's identity, a RACF user ID can be specified in the CICS IPCONN resource definition (if using an IPIC connection) or in the SESSIONS resource definition (if using an EXCI connection).

If it is necessary to run the CICS task with the requester's identity, implement a form of identity assertion by flowing either a RACF user ID to the CICS TG, or by implementing identity propagation.

When using identity assertion, a trust relationship should be established between the requester and the Gateway daemon, and between the daemon and CICS server (see "Trust" on page 162).

Integrity and confidentiality

Consider the following integrity and confidentiality related question. Does the integrity and privacy of the data warrant protection?

If messages do not contain critical data, or if the messages are only transmitted within an internal secure network, then it might be reasonable to flow messages in the clear.

If messages need to be protected, configure a secure connection between the requester and the Gateway daemon using a VPN or an SSL/TLS connection.

Note: SSL/TLS is not supported for ECI Version 2 clients.

Auditing

Consider the following auditing-related questions:

- Does the request need to be audited?

  Typical information that needs to be captured is the RACF user ID used for running the CICS task, the operation, and the nature of the data update.

- Do you need an audit log that contains the original distributed identity?

  Consider z/OS identity propagation support, which provides for identity assertion, control, and auditing as part of the built-in processing within CICS, RACF, and CICS TG.

Performance

The CICS TG is one of the best-performing connectivity options, especially for short-to-medium length messages. However, there is always a performance cost associated with security, so consider the following performance-related questions.

- Where is authentication done and how many times?

  The cost of authentication in CICS TG is dependent on whether you are using user ID and password authentication or X.509 certificates.

  If authentication is done by a WebSphere Application Server, the JCA resource adapter provided by the CICS TG can assert the original user's identity to CICS. This avoids the overhead of authenticating multiple times.

- Will you use SSL/TLS?

  When using SSL/TLS, it is particularly important to have an efficient connection-pooling mechanism, because otherwise, a significant proportion of the time, from making the connection to receiving the result from CICS and closing the connection, can be in the SSL handshaking. The JCA connection-pooling mechanism mitigates this overhead by allowing connections to be pooled by the WebSphere Application Server pool manager so that SSL handshaking for each request is not required.

- Are you using hardware cryptographic devices?

  Cryptographic hardware and ICSF is required to maximize performance when the Gateway daemon is configured to use SSL/TLS.

  You need to configure the Gateway daemon to use hardware cryptography.

  Important: The default with the CICS TG for z/OS is for hardware cryptography not to be used.

- What cipher suite are you using?

  To utilize hardware cryptography, the chosen cipher suite algorithm must be available in hardware.

6.8 WOLA

In a z/OS-only topology a JEE application running in a WebSphere Application Server can connect to CICS using the WebSphere Optimized Local Adapters (WOLA) JCA adapter. In this scenario, because both WebSphere and CICS are running on the same LPAR, a local cross-memory mechanism is used for connecting the two servers. This has certain security advantages:

- The application server and CICS are able to share the same RACF user registry for authentication and authorization checks.
- The application server and CICS are installed in the same MVS LPAR and, therefore, the connection between the servers is inherently more secure.
- It allows the identity associated with the WebSphere thread to be asserted into CICS so that the CICS task runs with the same identity. This form of identity assertion is known as *thread identity support*.

6.8.1 Thread identity support

WOLA is bi-directional and therefore also can be used to connect from CICS to a WebSphere Application Server running on z/OS (WAS z/OS). Thread identity support is available in both directions. Figure 6-11 shows the outbound and inbound WOLA integration scenarios and shows for each case whether thread identity support is enabled.

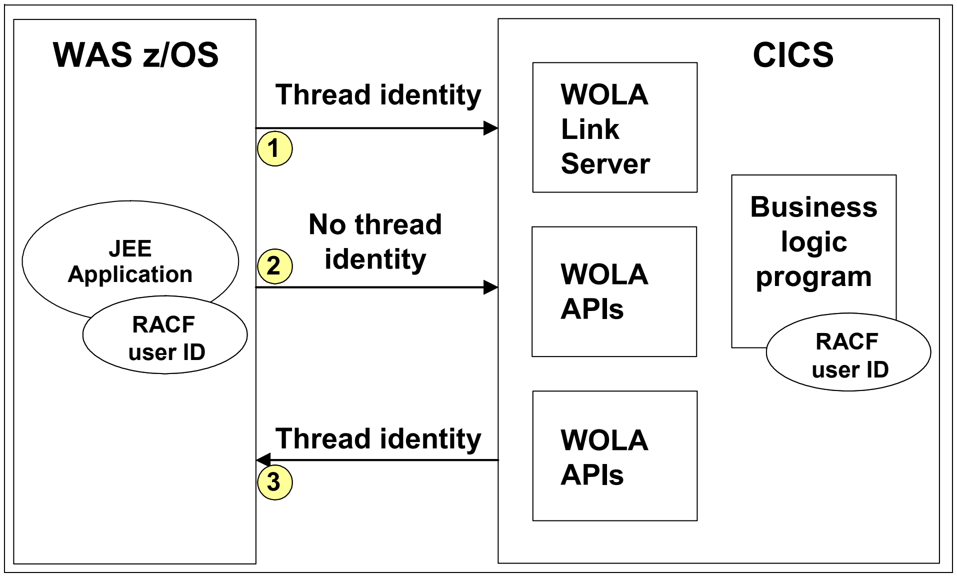


Figure 6-11 WOLA security options

Figure 6-11 shows the following scenarios:

1. Outbound from WAS z/OS using the WOLA Link Server task in CICS

   For receiving requests in CICS and processing them with the WOLA Link server task (BBO$), you can indicate when you start the link server that you want to have the link server assert the propagated WebSphere thread identity to the CICS task that runs the target business logic program. This is done with the `SEC=Y` parameter on the CICS BBOC control transaction.

   Surrogate security checks can also be enabled to confirm that the user ID associated with the WOLA link server task has the appropriate authority to start the CICS task with the WebSphere thread identity.

2. Outbound from WAS z/OS using the WOLA APIs in CICS

   For greater performance you can bypass the WOLA Link Server task. In this case, a CICS program uses the WOLA APIs directly. Thread identity assertion is not performed, and the user ID of the task that issues the WOLA APIs is also used for the task that runs the business logic program.

3. Inbound to WAS z/OS using the WOLA APIs in CICS

   Inbound requests to WAS z/OS run under the identity that is asserted in the EJB container, and optionally this identity can be the user ID that is being used for the calling CICS task. This requires the identity to have READ access to the WAS z/OS CBIND class. This behavior is controlled by the WebSphere environment variable **`ola_cicsuser_identity_propagate`**.

Note: Thread identity support is also available when using the CICS TG JCA resource adapter in local mode. See the ITSO Redbooks publication *CICS Transaction Gateway for z/OS Version 6.1*, SG24-7161.

6.8.2 Security considerations for WOLA

When requests pass between CICS and WAS z/OS, either inbound or outbound, take into account specific security considerations:

- For requests outbound from WAS z/OS, you need to determine whether the CICS started task should run with the WebSphere thread identity.
- For requests inbound to WAS z/OS, you need to determine whether the WebSphere application should run with the user ID of the calling CICS task or the CICS region user ID.
- If thread identity support is not required and performance is the highest priority:
  - For outbound from WAS z/OS to CICS, this is accomplished by specifying **`SEC=N`** and **`REU=Y`** on the WOLA Link Server **`BBOC START_SRVR`** command, or by using the WOLA APIs directly and not using the link server task.
  - For inbound from CICS to WAS z/OS, this is accomplished by setting the **`reg_flag_C2Wprop`** flag on the BBOA1REG API.
- You do not need to consider encryption of requests from WAS z/OS to CICS because cross-memory communication is used. However, implement a trust mechanism based on CICS surrogate security.

6.9 CICS web support

In this section we consider the security issues that arise when using CICS web support. Figure 6-12 on page 172 shows how CICS web support can be used for different types of clients:

- Web browser
- Atom feed reader
- Web service requester

The security capabilities provided with CICS web support can therefore be used in each of these different scenarios.

In Figure 6-12 CICS acts as an HTTP server. A web client sends an HTTP or HTTPS request to CICS. The security processing done by CICS is dependent on how you configure the TCPIPSERVICE and URIMAP, and whether an Analyzer program is used. CICS can also act as an HTTP client, in which case only the URIMAP is used to control security processing.

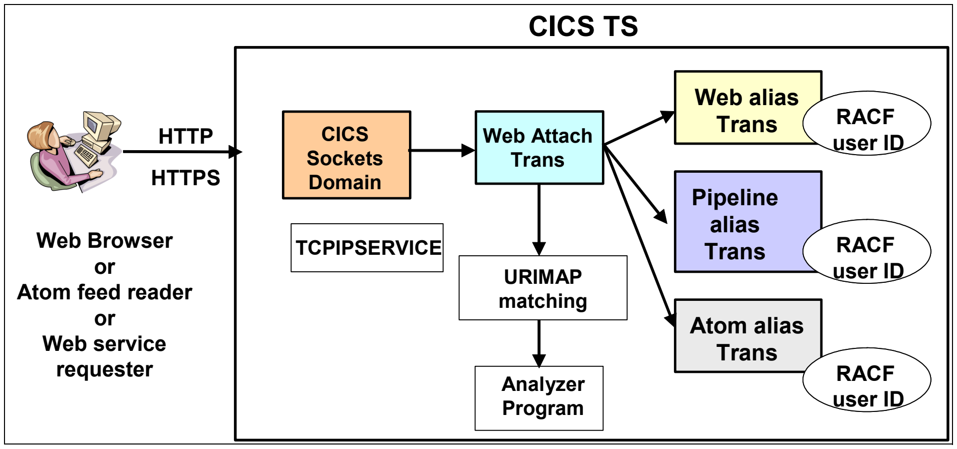


Figure 6-12 CICS web support

CICS web support security

The following security options are available with CICS web support:

- Basic authentication

  Basic authentication is an HTTP feature whereby the user ID and password are flowed over the network in a scrambled format that uses the Base64 encoding scheme. It is, however, easily unscrambled.

  To use HTTP basic authentication for an inbound web request to CICS, specify BASIC as the value of the **AUTHENTICATE** attribute of the TCPIPSERVICE definition.

  For outbound web requests from CICS, use a URIMAP definition to specify that basic authentication credentials can be captured by the global user exit, XWBAUTH. XWBAUTH then passes this information to CICS on request, and CICS sends the information in an HTTP authorization header.

- HTTPS

  HTTPS (HTTP over SSL/TLS) can be used for confidentiality, data integrity, and optionally for X.509 certificate authentication. HTTPS has the following advantages:

  – It provides a fast and secure transport.

  – It provides for authentication using a client X.509 certificate.

  – It provides integrity for the data passed between the HTTP client and the HTTP server.

  – It provides confidentiality for the data by using efficient secret key cryptography.

  – It can be used with hardware cryptographic devices that can significantly reduce the cost of SSL handshakes and data encryption.

  – It is mature and similarly implemented by most vendors, and therefore, is subject to few interoperability problems.

  HTTPS connections will automatically use the TLS 1.0 protocol, unless the client specifically requires SSL 3.0.

  CICS uses System SSL to support SSL/TLS, which in turn makes use of ICSF services and hardware cryptographic if available.

  Important: The use of cryptographic hardware and ICSF is strongly recommended to maximize performance when CICS is configured to use SSL/TLS.

  To use HTTPS for an inbound web request to CICS, specify YES as the value of the `SSL` attribute of the TCPIPSERVICE definition. Specify CLIENTAUTH if you also want the client to send a client certificate.

  To use HTTPS for an outbound web request from CICS, specify HTTPS as the protocol in the target URI. If the remote server requests a client certificate, then the default CICS certificate will be sent unless a URIMAP is specified on the `EXEC CICS WEB OPEN` command, in which case the certificate named in the URIMAP will be sent.

- Specifying a user ID in the URIMAP

  The `USERID` attribute of the URIMAP resource specifies the user ID to be used for the attached alias task. This user ID will apply to all inbound requests that match the SCHEME, HOST, and PATH specified in the URIMAP. This should only be used when the specific user does not need to be authenticated, but a user ID other than the CICS default user ID is required to authorize access to the associated resources.

An example of when this might be used is if the CICS default user ID is not authorized to run any alias transactions. If the real user is to be authenticated using an HTML forms-based dialog, then an alias transaction is required. A special user ID can be set up to allow a specific alias transaction and associated programs to be run before the real user ID is established.

- Analyzer program

  In most cases an analyzer program is not required. For example, it is not required if a suitable URIMAP definition is used.

  However, an analyzer program can use any information in the incoming HTTP request, for example, information obtained using the **`EXEC CICS WEB`** and **`TCPIP API`** to determine what user ID should be used for the alias task.

  The analyzer can also be used to specify that the user must supply their her ID and password. This can be done via HTTP Basic Authentication or a HTML forms-based dialog.

  Note: You should write or customize an analyzer program to authenticate the user only if the other methods of authentication are unsuitable. The analyzer program can perform other functions though, for example, it is a good place to write an audit log of web access to your CICS region.

Security considerations for CICS web support

Figure 6-13 shows a CICS web support security scenario in which a web client makes a call to CICS.

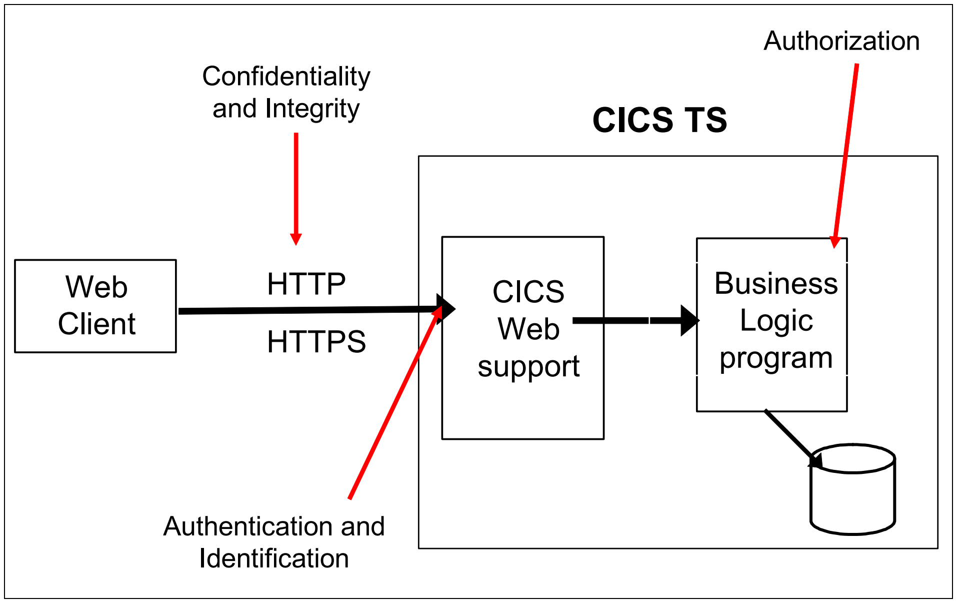


Figure 6-13 CICS web support security questions

The sections contain lists of questions and comments that will help you to choose between the different options available for securing the CICS web support scenario shown in Figure 6-13.

Authentication

Consider the following authentication-related questions:

- Does the web client need to authenticate?

  This can be decided for specific requests rather than there being a general rule for the application. It might be appropriate to run read-only requests using a generic user ID, which can be specified in a URIMAP.

- What authentication mechanism will be used?

  HTTP basic authentication is popular and easy to configure. However, the HTTP basic authentication scheme can only be considered a secure means of authentication when the connection between the web client and the CICS region is secure. If the connection is insecure, the scheme does not provide sufficient security to prevent unauthorized users from discovering and using the authentication information for a server. If there is a possibility of a password being intercepted, basic authentication should be used in combination with HTTPS, so that SSL encryption is used to protect the user ID and password information.

SSL client authentication requires more setup and is more expensive. Consider ways of optimizing performance, such as persistent TCP/IP connections and SSL session ID reuse. These optimizations mean that SSL handshaking can be avoided for requests following the initial handshake. To utilize hardware cryptography, the chosen cipher suite algorithm must be available in hardware.

For outbound web requests from CICS, by default CICS closes the connection after an application has finished using it. In CICS TS V4.2, you can set up connection pooling to reuse the connection to the same host and port. This is an important performance consideration if using HTTPS for outbound web requests.

Identification

Consider the following identification-related question. How will you assign the RACF user ID for running the CICS business logic program?

- It can come from the authentication credentials.
- It can be hard-coded in a CICS URIMAP.
- It can be extracted from the HTTP message by an analyzer program.

Authorization

Consider the following authorization-related question. Does the CICS task need to run with the web client's identity?

If it is not necessary to run the CICS task with the client's identity. A RACF user ID can be specified in a URIMAP.

Integrity and confidentiality

Consider the following confidentiality and integrity related question. Does the integrity and privacy of the data warrant protection?

HTTPS can be used to provide integrity and confidentiality of the data during the transmission.

6.10 WebSphere MQ

WebSphere MQ (WMQ) is a popular choice for integrating CICS applications with applications running in other servers. WebSphere MQ has a rich set of security mechanisms, including channel, connection, command, and resource security checking.

From a CICS integration perspective, the most relevant security options for consideration are as follows:

- User identities in messages

  The message descriptor (MQMD) structure contains the control information that accompanies the application data when a message travels between the sending and receiving applications. The structure is an input/output parameter on the `MQGET`, `MQPUT`, and `MQPUT1` calls. An *identity context* is part of the MQMD that contains identity-related information such as the *User Identifier*.

  The User Identifier field can be set by the application, or on z/OS more typically it is set by the environment, for example, CICS. For an MQPUT from a CICS application, this can be the CICS task user ID.

- CICS DPL bridge

  When using the CICS DPL bridge, different bridge security options can be configured, allowing for authentication of the bridge client or assertion of the User Identifier in the MQMD. An additional field (MQCIH.Authenticator) in the CICS bridge header (MQCIH) structure can be set to the password that is to be associated with the user ID in the MQMD.UserIdentifier field. Together, the values are used by RACF to determine whether the user is authorized to link to the DPL program.

  The level of authentication that you can use with the DPL bridge is as follows:

LOCAL	This level is the default. The bridge task and the CICS programs that are run by the bridge task are started with the CICS default user ID.
IDENTIFY	The bridge task is started with the user ID specified in the message (MQMD). CICS programs run by the bridge run with the user ID from the MQMD. There is no password checking. The user ID is treated as trusted.
VERIFY_UOW	The bridge monitor checks the user ID (in the MQMD) and password (in the MQCIH) before starting the bridge task. CICS programs run by the bridge run with the user ID extracted from the MQMD. If the user ID or password is invalid, the request fails with return code MQCRC_SECURITY_ERROR. Subsequent messages processed by this transaction are not checked.
VERIFY_ALL	This is the same as VERIFY_UOW except that the bridge task checks the user ID and password in every message.

- Securing access to WebSphere MQ resources

  As with CICS, RACF can be used to secure access to WMQ resources. WMQ checks for certain RACF classes and profiles within those classes to determine what security checks to perform.

  The RACF class MQADMIN is used to activate security for WMQ resources. Because RACF classes are shared across a Sysplex, there is a mechanism for turning off WMQ security, for example, in a test system.

  WMQ uses RACF profiles called *switch profiles* to toggle security at lower levels than the sysplex. Switch profiles contain the name of an individual queue manager or queue sharing group:

  - If security for the particular queue manager is not required, then WMQ sets its internal subsystem security switch off and performs no security checks.
  - If security checking is activated for a queue manager, WMQ checks the existence of specific RACF classes (MQCONN, MQCMDS, MQQUEUE, MQPROC, and MQNLIST) to control what resource level checks are performed. For example, MQCONN controls connection security (can CICS connect to this queue manager?) and MQQUEUE controls queue resource security (which queues can CICS access and how?).

  The hlq.RESLEVEL profile is defined in the MQADMIN class, and it controls how many user IDs are subject to API security checks. CICS can pass two user IDs to the queue manager:

  - The user ID associated with the CICS address space
  - The user ID associated with the CICS task

- SSL/TLS

  SSL/TLS can be used to secure messages that are transported between queue managers. You specify the cryptographic algorithms to be used by supplying a *CipherSpec* as part of the channel definition.

- WebSphere MQ Advanced Message Security

  The SSL/TLS support provided by WebSphere MQ provides security for messages while they are in transmit but not when they are held on message queues. WebSphere MQ Advanced Message Security (WMQ AMS) extends this support by providing *end-to-end* data protection of messages.

  Although the SSL/TLS support provided by WebSphere MQ facilitates the encryption and decryption of message data transmitted between two queue managers, or a client application and a queue manager, message data remains unencrypted when it resides on message queues. In addition, WebSphere MQ does not facilitate the signing of message data to ensure the identity of its author, or guarantee that message data has not been modified while on a queue or in transit between queues.

WebSphere MQ Advanced Message Security (WMQ AMS) expands WebSphere MQ security services to provide data signing and encryption at the message level. The expanded services guarantees that message data has not been modified between when it is originally placed on a queue and when it is retrieved. In addition, WMQ AMS verifies that a sender of message data is authorized to place messages on a target queue.

Figure 6-14 shows how WMQ AMS works.

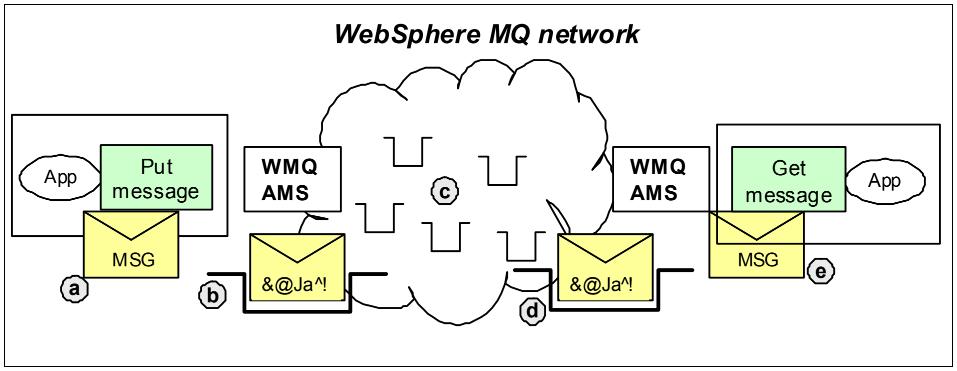


Figure 6-14 WebSphere MQ Advanced Message Security

This is the sequence of events shown in Figure 6-14:

a. The sender application uses the MQPUT API to put a message to a queue.

b. The MQPUT call is intercepted by a security exit that passes control to the WMQ AMS client interceptor, which manages pre-processing and post-processing for MQI calls. For example, if the target queue has a policy to sign and encrypt data messages, the WMQ AMS client interceptor signs and encrypts the message, then issues the actual call to WebSphere MQ to put the modified message to the target queue.

c. The signed and encrypted message is transmitted across the WebSphere MQ network.

d. The receiver application uses the MQGET API to get the message from a queue.

e. The WMQ AMS client interceptor performs signature checking and decryption as specified by the queue's data-protection policy, and then returns the original message to the calling application.

Note: When a CICS application issues a put to or get from a WMQ AMS protected queue, WMQ AMS uses a certificate associated with the CICS task user ID to sign, or sign and encrypt, according to the queue's policy.

For more information about WMQ AMS refer to the information center:

http://publib.boulder.ibm.com/infocenter/mqams/v7r0m1/index.jsp

Security considerations for WebSphere MQ

Figure 6-16 on page 183 shows a typical WebSphere MQ implementation.

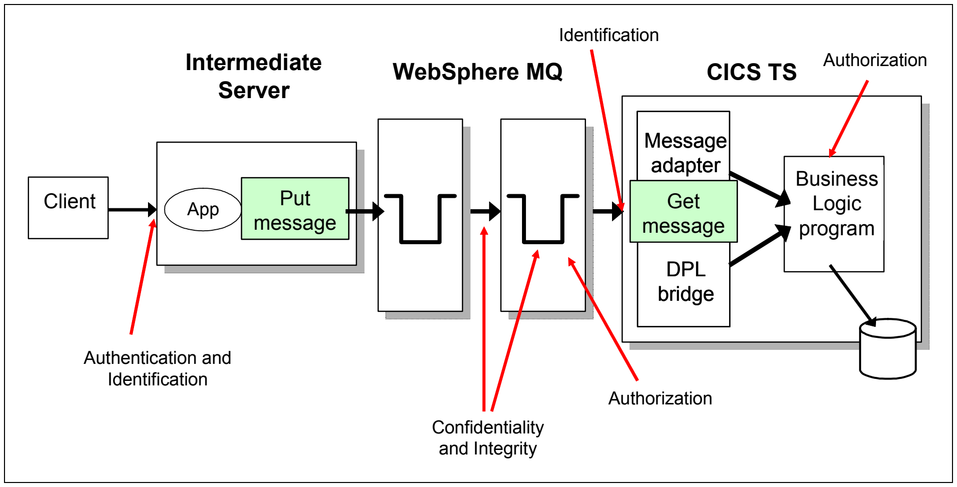


Figure 6-15 WebSphere MQ security questions

The following sections provide lists of questions and comments will help you to choose between the options available for securing the WMQ requests to CICS (Figure 6-15).

Authentication

Consider the following authentication-related question. How does the WMQ client authenticate?

Typically, authentication of the originating user has been done in advance, and it is a question of asserting a user identifier as part of the identity context in the message descriptor (MQMD).

User ID and password authentication can be enabled when using the DPL bridge.

Identification

Consider the following identification-related question. How will you assign the RACF user ID for running the CICS business logic program?

- It might come from the authentication credentials presented by a DPL bridge client.
- It might be asserted as a RACF user ID by a WMQ client.
- It might be allowed to default to the CICS default user ID.

Authorization

Consider the following authorization-related question. Does the CICS task need to run with the requester's identity?

If it is necessary to run the CICS task with the requester's identity, implement a form of identity assertion by flowing a RACF user ID in the MQMD.

When using identity assertion with the DPL bridge, the user ID of the bridge monitor must have surrogate authority for all the user IDs used in request messages.

Integrity and confidentiality

Does the integrity and privacy of the data warrant protection?

You can use SSL/TLS to secure messages that are transported using WMQ. In addition, you can also consider the use of WMQ AMS if messages require encryption while held in queues, or if message signing is required.

6.11 CICS sockets

CICS sockets are normally used when the programmer needs close control over the TCP/IP communication between the client and CICS application. Inherently, this form of communication does not benefit from all of the CICS-supplied access security mechanisms that are available with other integration options.

6.11.1 Using AT-TLS

The z/OS Communications Server TCP/IP stack provides Application Transparent Transport Layer Security (AT-TLS). This allows socket applications that use the TCP protocol to transparently use SSL/TLS to communicate with partners in the network. CICS sockets enabled applications can take advantage of this support.

To enable support for AT-TLS for a CICS sockets application, you need to create an AT-TLS policy configuration that matches the configuration of the CICS Sockets Listener transaction that uses it:

- The AT-TLS policy configuration includes settings that define whether the application is a listener or a client, the IP addresses, and the ports that are used for communication, and whether client authentication is required.
- The CICS Listener parameters include settings that define the transaction identifier, port, and whether the RACF user ID associated with the client's certificate is retrieved (GETTID option).

  When the GETTID option is configured, the Listener waits for the TLS handshake to complete on the accepted connection and then checks to see whether an associated user ID is present. A user ID is present when client authentication is defined in AT-TLS policy, the client passed in a certificate, and the certificate was registered with RACF with an associated user ID. This user ID is passed into the Listener security exit, if one is configured.

6.11.2 Listener security exit

The CICS sockets interface provides a *security exit* that can be used to authenticate socket clients. The Sockets Listener transaction (CSKL) links to the security exit before starting the child server task. The security exit decides whether to allow the Listener to start the server transaction based on the information it is passed, which includes this information:

Transaction ID	Transaction requested by client
Data area	User data received from client
Address	IP address of client
Socket	Socket descriptor

Note: It is entirely up to the security exit as to the criteria that it uses to permit or deny the request.

6.11.3 Security considerations for CICS sockets

Figure 6-16 shows a typical CICS sockets implementation.

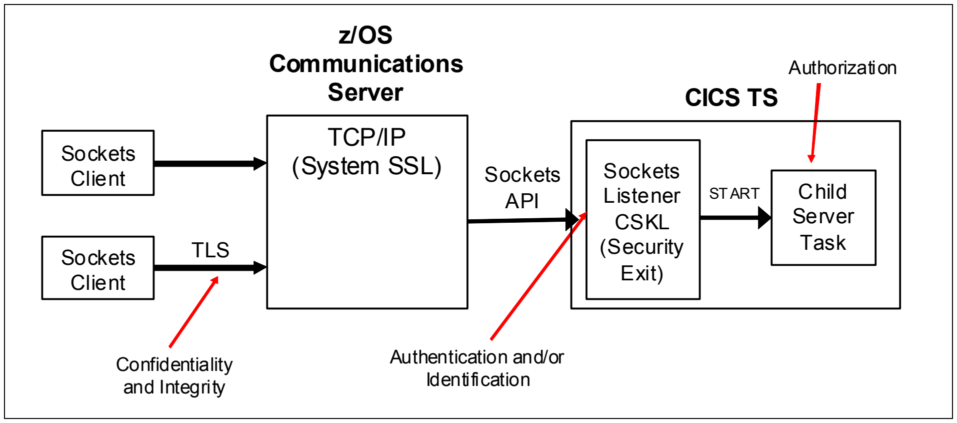

z/OS Communications Server
TCP/IP (System SSL)
Sockets Client
Sockets Client
TLS
Sockets API
CICS TS
Authorization
Sockets Listener CSKL (Security Exit)
START
Child Server Task
Confidentiality and Integrity
Authentication and/or Identification

Figure 6-16 CICS sockets security questions

The following list of questions and comments will help you to choose between the different options available for securing the socket requests to CICS shown in Figure 6-16 on page 183.

Authentication

Consider the following authentication-related question. How does the sockets client authenticate?

A user ID and password can be included in the data sent by the client. The security exit can then verify the user's credentials using the `CICS VERIFY USER`.

SSL client authentication can be enabled using AT-TLS.

Identification

Consider the following identification-related question. How will you assign the RACF user ID for running the CICS business logic program?

A user ID can be included in the data sent by the client. The Child Server transaction that runs the business logic program can be started with the user ID that is passed by the sockets client.

When SSL client authentication is used, the user ID that is associated with the client's certificate can be retrieved.

Authorization

Consider the following authorization-related question. Does the CICS task need to run with the requester's identity?

A user ID used in a basic authentication, or a user ID that is associated with the client's certificate, can be used to start the child server task.

You can also implement a form of identity assertion by flowing a RACF user ID in the data sent by the sockets client. When using identity assertion, a trust relationship should be established between the sockets client and CICS.

Integrity and confidentiality

Does the integrity and privacy of the data warrant protection?

If messages need to be protected, a CICS sockets enabled application can take advantage of the AT-TLS support provided by z/OS Communications Server. As shown in Figure 6-16 on page 183, the TLS encryption is done by TCP/IP on behalf of CICS.

7

Transactional scope

CICS TS is the predominant transaction processing system in use in today's IT systems. Therefore, any service requester that needs to connect to and utilize information from within CICS TS will most likely need to consider the transactional scope of such calls.

When considering the transactional scope of your new service requester, you need to think about the following key issues:

- Do any of the programs that you invoke within CICS TS perform work on recoverable CICS TS resources, such as VSAM files or DB2 tables?
- Do you need to ensure that multiple calls to the same CICS program are handled as a single recoverable unit?
- Do you need to coordinate recoverable work within CICS TS with work that is performed on other recoverable resources outside of CICS TS?
- Do you require CICS TS to control the runtime infrastructure to manage your transactional integrity or will your applications handle it themselves?

In this chapter, after a review of different transactional options, we discuss the main transactional considerations for each of the strategic CICS integration technologies.

7.1 Transactional objectives

A transaction is a unit of activity within which multiple updates to recoverable resources can be made atomic (that is, an indivisible *unit of work* (UOW)), such that all or none of the updates are made permanent.

A classic example of a transaction is the movement of money from one bank account to another. There are two operations:

- Deduct the money from the sending account.
- Add the money to the receiving account.

To avoid losing or gaining money is it important that both operations occur (the transaction commits) or that neither operation occurs (the transaction is rolled back). This can be achieved by performing both operations within the same unit of work, then committing or rolling back the unit of work.

Within a distributed transactional system, each distributed system is either referred to as a *resource manager* or a *transaction manager.* The transaction manager controls the outcome of the transaction (should it commit or roll back?) and is responsible for the recovery of its resources. It has to implement a recoverable logging mechanism in order to be able to coordinate multiple resource managers. The resource managers control access to recoverable resources, and as such have to implement the necessary network flows and logging procedures to provide transactional coordination.

CICS TS can be both a transaction manager and a resource manager.

When considering CICS integration, consider the following questions related to the transactionality of your architecture:

- Do you have recoverable resources that need to be transactional?
- Where are those resources accessed from?
- Where are the transactions managed from?

There are several approaches to managing transactions and there are several different transactional scopes. The intention of the remainder of this chapter is to enable you to take the following actions:

- Determine the transactional approach/scope that is required for your integration architecture.
- Use 7.4, "CICS Web services" on page 198 to understand which integration technologies support the transactional approach/scope that you decide upon.
- Refer to the sections about the integration technologies to gain further information about how they support the transactional approach/scope that you require.

Before considering the different transactional building blocks, it is important to be aware that CICS has traditionally used the term *transaction* in a different context from meaning a unit of work, so let us first look at CICS transactions and associated terminology.

In this chapter, we refer to a *CICS-transaction* as the work initiated in a CICS region and that runs as a CICS *task* under a four-character transaction ID (tranid). These tranids are static definitions that specify the initial program to be loaded and the properties of the CICS-transaction under which the program will run. They are defined in TRANSACTION definitions within the CICS resource definition online (RDO) database.

7.2 Transactional building blocks

In this section, we examine the different transactional building blocks that can be used to ensure that the overall architecture of the system has the appropriate transactional characteristics. We start with an overview of the scope of units of work within CICS TS, then consider transactional scopes that extend across a wider distributed environment.

7.2.1 Traditional CICS units of work

At task initiation, CICS TS implicitly starts a unit of work for all CICS transactions. This is usually the initial boundary of the transactional work to be undertaken. All updates to recoverable resources or requests to other transactional systems are now part of this unit of work, until either a synchronization point (syncpoint) is reached within the CICS program, or the CICS-Transaction finishes and the task terminates (Figure 7-1).

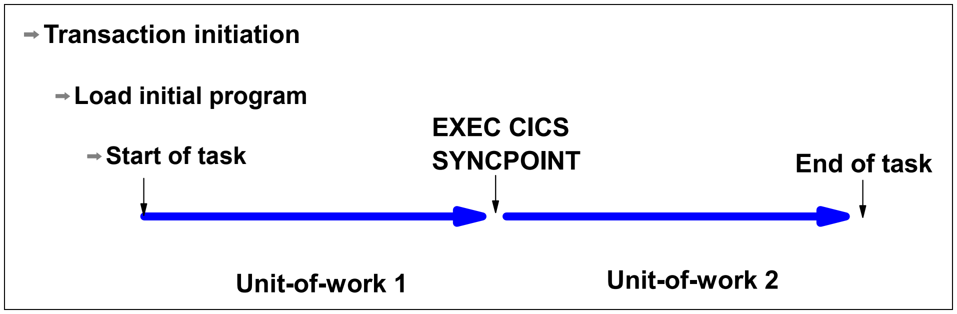


Figure 7-1 CICS TS synchronization points

In certain circumstances, such as when an inter-system distributed program link (DPL) request is made, the CICS-Transaction that is linked to can be coordinated by a remote CICS region. In the region that is linked to, a *mirror* task runs. The job of the mirror task is to handle the communication from the calling CICS region. On return to the calling CICS region, the mirror task remains suspended until the end of the transaction, when it is told to commit the unit of work. This is referred to as a *long-running mirror task* (Figure 7-2).

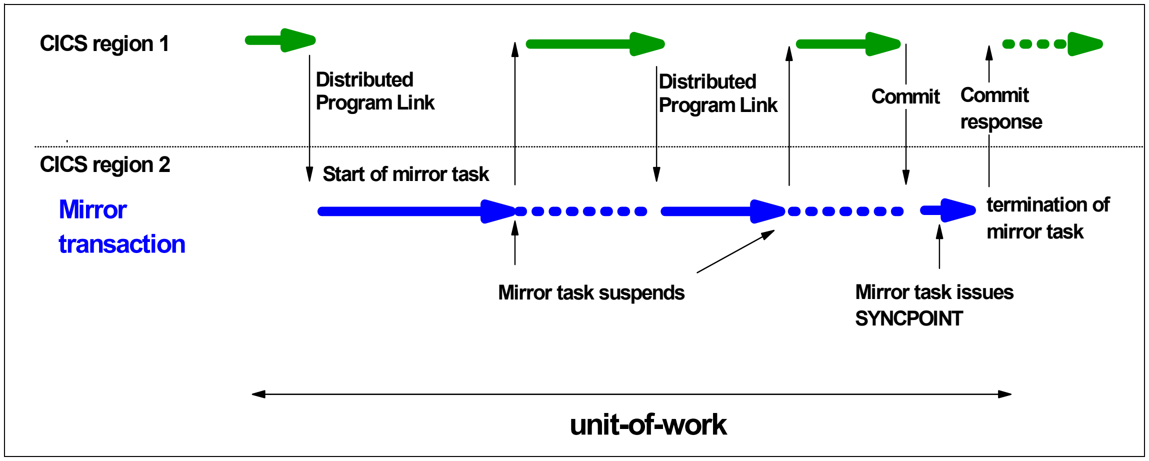


Figure 7-2 Link with long-running mirror task

Additionally, the converse situation is also possible; this is where the invoked CICS-Transaction runs in a separate transactional context to that of the invoking application. This is referred to as *running with sync-on-return*, which refers to the fact that the controlling mirror transaction in CICS issues a syncpoint on returning control to the calling application (Figure 7-3). The use of a sync-on-return type link also allows the called CICS program to issue **`EXEC CICS SYNCPOINT`** commands, because it is not subordinate to another transaction manager.

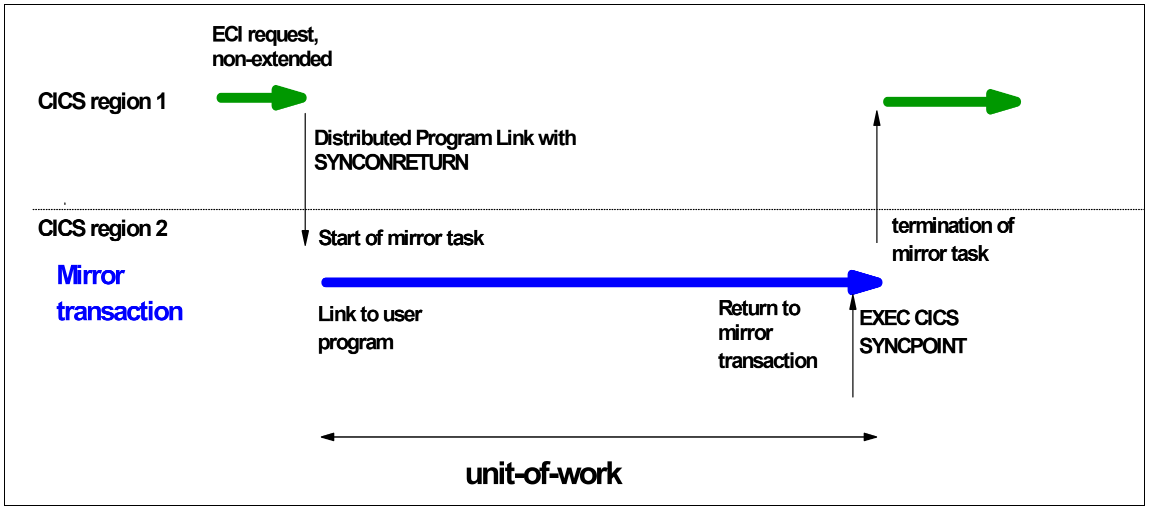

ECI request, non-extended
CICS region 1
Distributed Program Link with SYNCONRETURN
CICS region 2
Start of mirror task
termination of mirror task
Mirror transaction
Link to user program
Return to mirror transaction
EXEC CICS SYNCPOINT
unit-of-work

Figure 7-3 Link with SYNCONRETURN

Note: If you are considering integrating applications with existing CICS programs, it is important to understand the scope of units of work within those programs so that you can design and implement the right overall transactional architecture for your solution.

7.2.2 Extended logical units of work

Extended logical units of work (Extended LUW) extend the concept of allowing a remote CICS region to coordinate a transaction in another CICS region, as described in 7.2.1, “Traditional CICS units of work” on page 187. In an extended LUW, an invoking application that is external to CICS TS can coordinate a transaction in a CICS region.

The invoking application makes a call to a CICS region, which runs a program under a unit of work and returns, leaving the unit of work uncommitted. The invoking application can then make further calls to the same CICS region and CICS TS performs transactional resource updates under the same unit of work.

The invoking application controls whether the unit of work is committed or rolled back.

One of the technologies that offers this capability is CICS Transaction Gateway (CICS TG) (Figure 7-4).

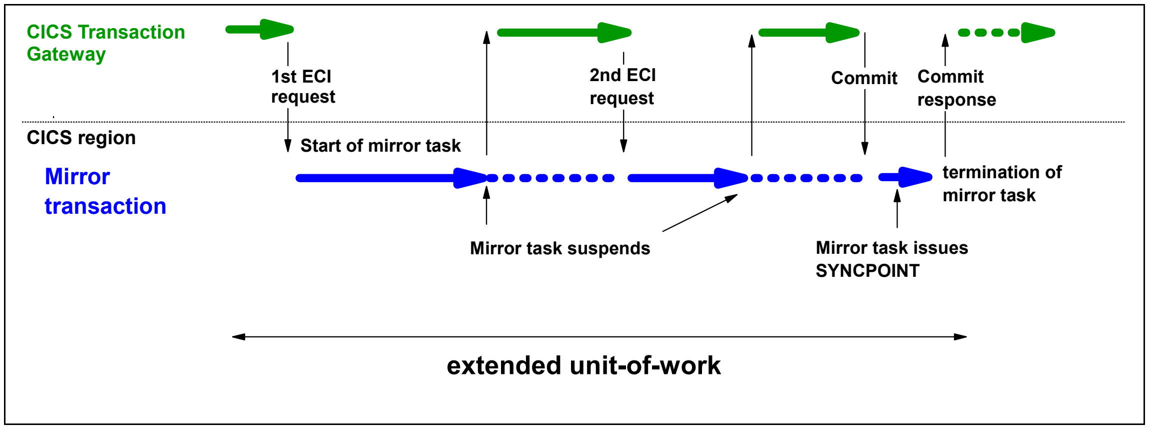


Figure 7-4 Extended logical unit of work

In Figure 7-4, CICS TG is shown using a communications protocol called the External Call Interface (ECI) to invoke a CICS program. On return from the CICS program, the mirror task suspends. A second ECI request is made from CICS TG, which is processed by the same mirror task and uses the same unit of work. On return from the program, the mirror task suspends again, until it is told by CICS TG to commit the unit of work, at which point it issues a SYNCPOINT.

Note: In cases where there is only one resource manager, the resource manager can be told to either commit or roll back. This is often referred to as one-phase commit.

7.2.3 Distributed units of work

Distributed units of work add value when transactional resource updates need to be synchronized across different systems. For example, a service requester might need to make a local update to a database within a unit of work, then invoke CICS TS to make a transactional update to a VSAM file within the same unit of work. To ensure that the updates across all the resource managers are all committed or are all rolled back, a protocol known as two-phase commit is used.

Note: Distributed units of work are also known as global units of work.

Two-phase commit protocol overview

An essential part of all transactional standards is the two-phase commit process. This is an architected set of flows that transaction managers use to ensure that all resource managers in a transaction can be reliably coordinated, irrespective of any failure. It is implemented by various transactional protocols, and the fundamental concepts are essentially the same. The following description summarizes the flows according to the XA specification. XA is a specification for distributed transaction processing, allowing two-phase commit processing.

From the point at which a distributed unit of work begins to the first phase of commit described below, the unit of work is known as being *in-flight*. While a unit of work is in-flight, CICS TS (or any other resource manager enlisted in the transaction) can choose to roll back the unit of work. For example, if CICS TS terminated abnormally, at restart it would automatically roll back the unit of work.

In the first phase (or stage 1) of commit processing, the transaction manager asks all the resource managers to prepare to commit recoverable resources (prepare). Each resource manager can vote either positively (*prepared*) or negatively (*rolled-back*). If a resource manager is to reply positively, it records stably the information that it needs to do so and replies *prepared*, and is then obliged to follow the eventual outcome of the transaction as determined at the next stage. The resource manager is now described as *indoubt,* because it has delegated eventual transaction control to the transaction manager.

In stage 2, providing that all the resource managers voted positively, the transaction manager replies to each resource manager with a commit flow. Upon receipt of the commit flow, the resource manager finalizes updates to recoverable resources and releases any locks held on the resources. The resource manager then responds with a final *committed* flow, which indicates to the transaction manager that it is no longer in doubt. If the final committed flow is not received by the transaction manager, the transaction manager must assume that the commit was also not received by the resource manager and must re-transmit the commit.

Figure 7-5 shows the two phases of the commit process: prepare, followed by commit (or roll back).

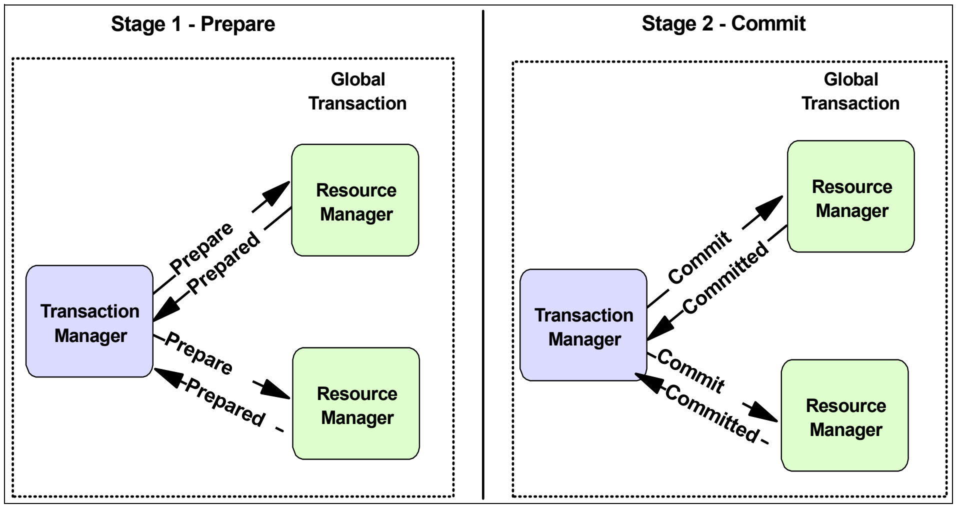


Figure 7-5 Two-phase commit

Note: In-doubt units of work cannot be unilaterally rolled back by one of the resource managers without the risk of a mixed outcome to the unit of work (where some resources are committed and some are backed out). This is an important consideration when choosing to use distributed units of work.

While the unit of work is prepared, resources will be locked. If the resource manager loses contact with the transaction manager, these resources could be locked for a period of time until contact is re-established. Locked resources can impact the running of a system, so when considering distributed units of work, it is important to consider the availability of the transaction manager, the reliability of the network connection, and the impact to the workloads in the event of a communications failure with the transaction manager.

Resource managers can often be configured to take a best guess at the outcome of a prepared unit of work, if the unit of work is not committed or rolled back by the transaction manager in a reasonable amount of time. This is known as a *heuristic decision*. It stops locks from being held too long, but creates the risk of a mixed outcome.

Optimizations for distributed units of work

Distributed units of work add additional protocol flows, so they should only be used when really needed. Sometimes there are optimizations that can be made by the systems involved in a distributed unit of work, which allows a one-phase commit to be used. These are discussed below.

Read only

If a resource manager has registered for a unit of work but has not performed any recoverable updates, when it is told to prepare, it can return a state of *read only*. This tells the transaction manager that the resource manager is not interested in the transaction, so the transaction manager does not need to send a commit or rollback request to the resource manager.

Last resource optimization

Although the two-phase commit process is usually a prerequisite to distributed transactional support, there are certain instances where a single-phase commit process can be sufficient. This is referred to as last resource optimization and is implemented by a variety of transaction managers. It essentially allows the commit decision to be delegated to the one-phase commit resource, allowing the one-phase commit to participate in a distributed unit of work with any number of two-phase commit capable resources (Figure 7-6).

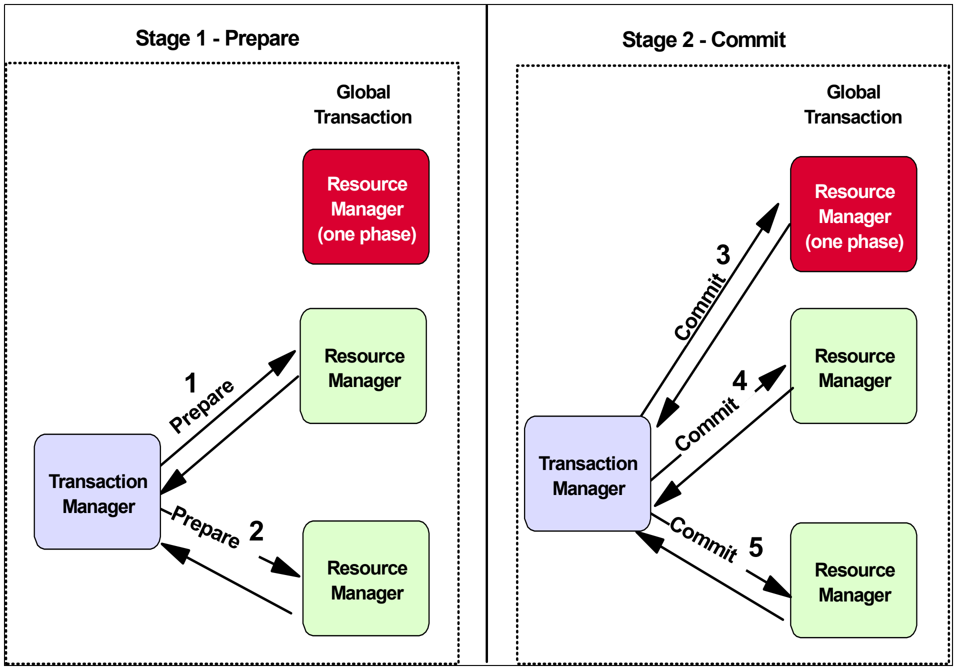


Figure 7-6 Last resource optimization

At transaction commit, the transaction manager first prepares the two-phase commit resource managers and, if this is successful, the one-phase commit-resource is then called to commit. The two-phase commit resources are then committed or rolled back depending on the response of the one-phase commit resource, effectively delegating transaction coordination to the one-phase commit resource.

Unlike a two-phase commit resource, there is no recovery from a communication failure with a one-phase commit resource. Such a communication failure during commit of the one-phase commit resource introduces the risk of a mixed outcome to the transaction. The two-phase commit resources are rolled back, but the outcome of the one-phase commit resource is unknown. It could have committed or rolled back. Applications must therefore be configured to accept the additional risk of such heuristic outcomes.

Last resource optimization is implemented within WebSphere Application Server as *Last Participant Support* and within CICS TS and APPC flows as *last agent optimization*.

Only agent

If there is only one resource manager that makes transactional updates within a distributed unit of work, then the two-phase commit process is not required. With only agent support, the transaction manager uses a one-phase commit in this instance (the prepare phase is dropped). The resource manager is told directly to commit or roll back.

7.2.4 Asynchronous messaging transactional model

In a synchronous model, the expectation is that the service requester and the service provider will both be available at the same time. An asynchronous model removes this requirement by adding an intermediary queue of work. The service requester puts a request message onto the queue. The service provider gets the request message from the queue. This pattern introduces a different transactional model, with two transactions typically used to ensure that the request message is delivered to the service provider. Figure 7-11 on page 214 shows an example of this processing.

In the asynchronous messaging transactional model:

1. The service requester puts a request message to a queue in the same unit of work as it performs local transactional resource updates. The service requester commits the unit of work.
2. The request message can now be consumed by CICS TS within a new unit of work. If CICS TS needs to make transactional updates, these are done within the same unit of work, which is then committed.

To ensure that the request message is delivered, define it to be persistent. It is the responsibility of the messaging product to ensure that the message can be recovered in the event of a failure.

Figure 7-11 on page 214 is an example of a one-way style of asynchronous messaging. The service requester does not expect a reply message. The assured delivery characteristics of the WebSphere MQ transport, together with the transactionality used in CICS TS to process the message, allows this model to work. The model can be extended such that CICS TS puts a reply message to another queue within the same unit of work as it read the request message and processed it. The reply message can then be consumed by the service requester or another consumer.

Pseudo-synchronous request reply

A common messaging pattern is pseudo-synchronous, where the service requester puts a request message and then issues a get, waiting for the reply message. In this model, you must decide whether to use persistent messages. If persistent messages are not used, it is possible that the service requester will not receive a reply, as the request or the reply message can be lost in the event of a failure. If persistent messages are used, a reply should be received, but the length of time before it is received is unknown and the requester might time out waiting for it. In these circumstances you could consider idempotent requests, as described in 7.2.6, "Idempotent requests" on page 196.

Conversational request reply

Another pseudo-synchronous pattern is conversational, where the requester sends a request, gets the reply back, and sends one or more further requests based on the information in the reply. CICS TS can enable conversational requests to make updates to transactional resources within the same unit of work, for example, an extended logical unit of work. In this case, the request messages consumed by CICS TS and the response messages put by CICS TS are performed outside of the extended logical unit of work.

7.2.5 Compensating transactions

A compensating transaction is a group of operations that undoes the effects of a previously committed transaction. There are many circumstances where compensating transactions might play a role:

- They might be used to restore consistency after an unrecoverable failure that prevented a distributed unit of work from normal completion.
- A resource manager might have been left in-doubt because it did not receive a reply in the second stage of the two-phase commit process. If so, it might have taken a heuristic decision about the probable outcome of the

transaction, and so some participants might have committed while others did not.

- When one of the distributed unit of work participants is a non-transactional resource manager. If such a transaction performs a rollback, its non-transactional participant might need to be rolled back via the compensating transaction.
- In certain business transaction scenarios, especially ones that span several systems, maintaining long-lived locks and restricting data access for extended periods of time might not be acceptable options. In these situations, it might not be desirable to map business transactions into single distributed units of work, but split them into more manageable units of work and provide compensating transactions to perform rollbacks.

Note: It is important to stress that an application that depends on compensating transactions must have extra logic to deal with failures and the possibility that further updates are made to the resource in between the original committed transaction and the undo transaction. Otherwise, the data might be left in an inconsistent state. For these reasons, their usage should be carefully evaluated.

7.2.6 Idempotent requests

An idempotent request is one that will only ever be processed once by the service provider, no matter how many times it is sent from the service requester.

It is sometimes possible for a response message not to be delivered back to an invoking application:

- The request message could be lost en-route to the service provider.
- The service provider could fail after committing transactional resource updates, but before sending the response.
- A response message could be lost en-route back to the service requester.
- The invoking application could fail after issuing a request and before receiving the response.

In these cases, the service requester does not know whether the request was processed. You could choose to resend the request, but if the request had already been processed, then the same update might be made a second time, which might not be appropriate.

An idempotent request contains a unqiue identifier. If the request is sent to the service provider and the service provider has already processed a request with that unique identifier, it does not process it a second time. The service provider replies with the same data that it would have returned on the original response to the request. Using idempotent requests does put an emphasis on the service requester to create/obtain a unique identifier and to persist that identifier, if you want the service requester to be able to resend a request after recovering from a failure. In CICS TS, a service requester could use a *named counter server* to obtain a unqiue ID, or as an example it could be a combination of fields, such as a customer account number and a timestamp.

This capability can be used in any situation where the service requester is not determining the commit/rollback decision. If the service requester is determining the commit/rollback decision, it can issue a rollback of the unit of work before re-sending the request.

7.3 Technology comparison table

This section describes which transactional building blocks are supported through the various access technologies that CICS TS supports. 7.4, "CICS Web services" on page 198 provides this comparison in table form. Refer to the technology-specific sections that follow Table 7-1 for a more detailed explanation.

Note: As compensating transactions and idempotent requests are technology agnostic, they are not included in the table, but can be considered for all technologies.

Table 7-1 Transactional building blocks: Technology comparison table

	CICS Web services	**CICS TG for z/OS**	**WOLA**	**CICS web support**	**WebSphere MQ**	**CICS sockets**
Traditional CICS units of work	Supported	Supported	Supported	Supported	Supported	Supported
Extended logical units of work	Not supported	Supported when using JCA and ECI v2	Not supported	Not supported	Supported	Not supported

	CICS Web services	CICS TG for z/OS	WOLA	CICS web support	WebSphere MQ	CICS sockets
Using distributed units of work to coordinate service requester and service provider updates	Supported when using WS-Atomic Transactions	Supported when using JCA	Supported when using JCA from WebSphere Application Server for z/OS on same LPAR	Not supported	Not supported	Not supported
Optimizations for distributed units of work	Not supported	Supported when using WebSphere Application Server	Supported	Not supported	Not supported	Not supported
Asynchronous messaging transactional model	Supported when using WebSphere MQ	Not supported	Not supported	Not supported	Supported	Not supported

7.4 CICS Web services

Web services provide an open standards based communications method and enable interoperability between a wide range of service requesters and service providers. CICS TS can be both a service requester and a service provider. As a service requester, through the use of distributed units of work, CICS TS can coordinate transactional updates across service providers, extending the types of transactional resources that CICS TS can manage to any with a web service interface. This is a very powerful feature. As a service provider, distributed units of work enable updates in CICS TS to be coordinated by an external transaction manager with updates to resources external to CICS TS.

In the next section, we examine which of the building blocks can be used with CICS Web service support.

7.4.1 Supported building blocks for CICS Web services

Figure 7-7 shows transactional scopes supported by CICS Web services.

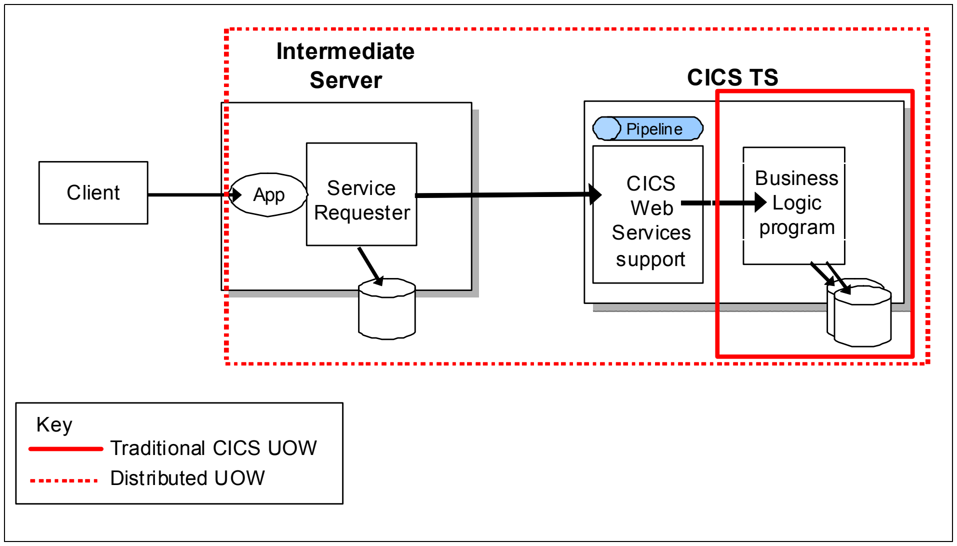


Figure 7-7 Transactional scopes supported by CICS Web services

Traditional CICS units of work

The default transactional behavior of web services is undefined by the web service specifications. For inbound web service requests into CICS TS, CICS TS creates a unit of work under which to process recoverable updates, as described in 7.2, “Transactional building blocks” on page 187. This enables updates to multiple CICS resources, spanning multiple CICS regions, to be all committed or all rolled back. For outbound web service requests from CICS TS, you need to refer to the documentation of the service provider to understand whether multiple resource updates are performed within a unit of work or outside of a unit of work.

Distributed units of work

CICS TS provides support for distributed units of work over web services through implementation of a web services standard called *WS-Atomic Transactions (WS-AT)*. WS-AT implements the two-phase commit protocol described in 7.2.3, “Distributed units of work” on page 190. The WS-AT protocol messages are themselves web services messages, and so are a form of XML.

The WS-AT specification can be found here:

http://www-128.ibm.com/developerworks/library/specification/ws-tx

A thorough overview of CICS TS support for WS-AT is available in the IBM Redbooks publication *Implementing CICS Web Services,* SG24-7206.

When acting as a web service requester, CICS TS can coordinate transactional updates across other systems that support the WS-AT protocol. When acting as a web service provider, CICS TS can participate in distributed transactions that are coordinated by an external transaction manager. Both the web service requester and the web service provider must support the WS-AT protocol in order to interoperate when distributed units of work are required.

Asynchronous messaging transactional model

CICS TS supports web services over the WebSphere MQ transport. Web service requests can be one-way or request/response.

For inbound web service requests, the asynchronous messaging transactional model enables assured delivery of the web service request message into CICS TS. Further information about configuring one-way web service requests over WebSphere MQ can be found in *CICS TS V4.2 Web Services Guide,* SC34-7191. For outbound web service requests, refer to the documentation of the service provider to understand whether messages are consumed under the same unit of work as transactional resource updates are made.

CICS TS requires a connection to a WebSphere MQ Queue Manager running on the same LPAR as the CICS region. For inbound requests, having got a message from the queue, in the event of an error, any transactional resource updates made by the business logic program are backed out, and the get of the MQ message is also backed out. The message becomes available on the queue again.

Pseudo-synchronous request reply

The pseudo-synchronous usage of WebSphere MQ can also be used for the web service request/response messaging pattern. In this pattern the requester is waiting for a response, and if there is a need for distributed units of work, the use of WS-Atomic Transactions can be considered. See “Distributed units of work” on page 199.

Note: There is little value in using persistent MQ messages if the WS-AT protocol is being used to support distributed units of work. In the rare event that a non-persistent message was lost, the distributed unit of work could be rolled back and the request sent again.

Compensating transactions

There is no specific support needed for writing compensating transactions within CICS TS. The user can write an additional CICS Web service that can be called to perform the business logic necessary to compensate for a previous transactional update.

Idempotent requests

If the decision is made not to use distributed units of work, then idempotent requests could be used to ensure that in the event of a response message not being received, a request message can be retransmitted without causing the same updates to be made twice at the service provider.

7.4.2 Transactional considerations for CICS Web services

The following list of questions and comments will help you to choose between the tranactional building blocks available for CICS Web services:

- Is the traditional CICS unit of work support good enough?

  If all the transactional resources that you need to coordinate are accessed from CICS TS, the traditional CICS unit of work support should meet your requirements. This will be the best-performing option.

- Do you need distributed units of work?

  If you need to coordinate updates to resources in CICS TS with updates to resources in other systems synchronously, distributed units of work could be the answer. Alternative solutions include compensating transactions, or using WebSphere MQ if an asynchronous model is appropriate.

- What is the overhead of using WS-Atomic Transactions to enable distributed units of work?

  The WS-Atomic Transactions protocol messages are web services messages and are a form of XML. There is a processing cost for parsing XML that should be taken into consideration. A typical (successful) transaction involving WS-AT will involve three request/response web services message pairs being sent between the transaction manager and each participating resource manager for each distributed unit of work (register/registerresponse, prepare/prepared, and commit/committed).

- Should you use web services for distributed units of work, or an alternative technology like CICS TG?

  Web services are based upon open standards. One of the key benefits that they provide is a wide level of interoperability with other systems. This interoperability is gained by using XML-based messages and keeping standards as simple as possible. Web services provide inbound and outbound capabilities.

CICS TG is a product designed specifically to connect environments such as Java EE application servers into CICS TS. It is highly optimized for these environments, and is the most commonly used connector into CICS TS.

- How will distributed units of work affect high availability?

  Considerations need to include deadlock avoidance and unit of work affinities. Refer to 8.3.3, “High-availability considerations for CICS Web services” on page 234 for further details.

- For inbound calls to CICS TS, do your CICS programs already make SYNCPOINT calls?

  If so, you will need to change those programs if you want the unit of work to be coordinated from the invoking application. If a CICS program attempts to issue an explicit SYNCPOINT when the CICS region that it is running in is not coordinating the transaction, the SYNCPOINT command fails with an error return code, which if unhandled causes a program abend. The CICS region can be told to allow syncpoints by invoking the CICS program with SYNCONRETURN, as described in 7.2.1, “Traditional CICS units of work” on page 187, but this means that the CICS updates are not coordinated by the invoking application.

7.5 CICS TG for z/OS

In this section, we look at the transactional scope options when using the CICS TG to connect into CICS TS. We consider the options when using the JCA resource adapter and also when connecting from an ECI v2 client.

7.5.1 JCA

One of the primary use cases of the CICS TG connector technology is to provide a highly optimized connection between a Java EE application server and CICS TS. In this section we look at the transactional capabilities that the CICS TG provides in this scenario.

The Java EE Connector Architecture (JCA) is part of the Java EE standard implemented by application servers such as WebSphere Application Server. The JCA specifies the system contracts for connection management, transaction management, and security management that exist between the application server and enterprise information systems (EIS) such as CICS TS (Figure 2-5 on page 27).

For transaction management, the resource adapter is required to implement one of the following contracts, as defined in the resource adapter's deployment descriptor (`ra.XML`):

- XAResource

  An XAResource is a transaction participant that is called during two-phase commit and that can influence the outcome of the transaction. Typically, an XAResource is implemented by a resource manager and is used to support the external coordination of the resource manager's transaction branch. Enlistment of an XAResource in a transaction is managed by the application server and is not a concern of the application.

- LocalTransaction

  A resource adapter that can participate in transactions that are local to the resource manager (one-phase commit), but that cannot participate in two-phase commit transactions (other than as an only agent or a last participant).

- NoTransaction

  A resource adapter with no transactional properties, that can participate in a transactional context but is not influenced by, and has no effect upon, the outcome of the transaction.

The CICS Transaction Gateway for z/OS supplies two resource adapters:

- ECI resource adapter

  The ECI resource adapter implements the LocalTransaction interface.

- ECI XA resource adapter

  The ECI XA resource adapter implements the XAResource interface and has full support for distributed units of work.

JCA support is provided in WebSphere Application Server within the web and EJB containers, both of which provide support for the JCA connection pooling mechanism and propagation of the transaction context from the Java EE component to a JCA interaction.

The web container provides limited transactional support, details of which can be found in the WebSphere Application Server documentation.

The EJB container is ideally suited to the deployment of transactional components and provides support for both container-managed transactions and bean-managed transactions. Container-managed transactions are the preferred mechanism, as this delegates transactional control to the application server, allowing the application developer to concentrate on developing the business logic, while still allowing the transactional properties of the application to be decided upon deployment. The key to transactional control with container-managed transactions is the EJB *transaction* attribute.

EJB transaction attribute

The transaction attribute is set in the assembly descriptor section of the EJB deployment descriptor (that is, the `ejb-jar.xml` file). This attribute is used by the EJB container to control under which circumstances a *global* transaction is started when a bean method is invoked.

This transaction attribute appears in the `<container-transaction>` section and is specified with the `<trans-attribute>` tag. For example, the following XML specifies that the `execute()` method on the `CICSTGTesterCCI` bean has the transaction attribute of `Required`:

```
<container-transaction>
   <method>
      <ejb-name>CICSTGTesterCCI</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>execute</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
```

The possible values for the transaction attribute are NotSupported, Required, RequiresNew, Supports, Mandatory, and Never, and their meanings are described in Table 7-2.

Table 7-2 EJB transaction attribute settings

Transaction attribute	Meaning	Resulting ECI JCA request	CICS TS mirror task
NotSupported	Bean method cannot execute within context of an OTS transaction	Non-extended LUW	SYNCONRETURN
Required	Bean method must execute within context of an OTS transaction	Extended LUW	Long running
RequiresNew	Bean method must execute within context of a new OTS transaction	Extended LUW	Long running
Supports	Bean method can execute with or without an OTS transaction context	Non-extended or extended LUW[a]	SYNCONRETURN or long running
Mandatory	Bean method must execute within context of client's OTS transaction	Extended LUW or Exception thrown [b]	Long running
Never	Bean method must not be invoked in context of an OTS transaction	Non-extended LUW	SYNCONRETURN

a. For *Supports* the result depends on the existing transactional context. The bean method is executed either under the caller's transaction context, and the ECI call will use an extended logical unit of work, or, if there is none present, then it executes under an unspecified transaction context.
b. For *Mandatory*, the bean method is executed under the caller's transactional context. If the caller does not supply a context (that is, there is no global transaction active), then the execute fails with an exception.

Generated code

There are a number of products available, such as WebSphere Integration Developer, that will automatically generate code that uses the JCA to invoke CICS TS applications. This additional level of abstraction means that the transactional settings mentioned in Table 7-2 on page 204 must be made through the tooling so that the auto-generated code contains the right transactional scope.

7.5.2 ECI v2

The CICS Transaction Gateway can also be used from unmanaged environments, such as Microsoft .NET or native C applications. In unmanaged environments, the emphasis is on the application to control transactionality, and the options are more limited than from a managed Java EE environment.

7.5.3, “Supported building blocks for CICS TG” on page 206 describes the options available and whether they are supported when using JCA or ECI v2.

7.5.3 Supported building blocks for CICS TG

Figure 7-8 shows different transactional scopes supported by CICS TG.

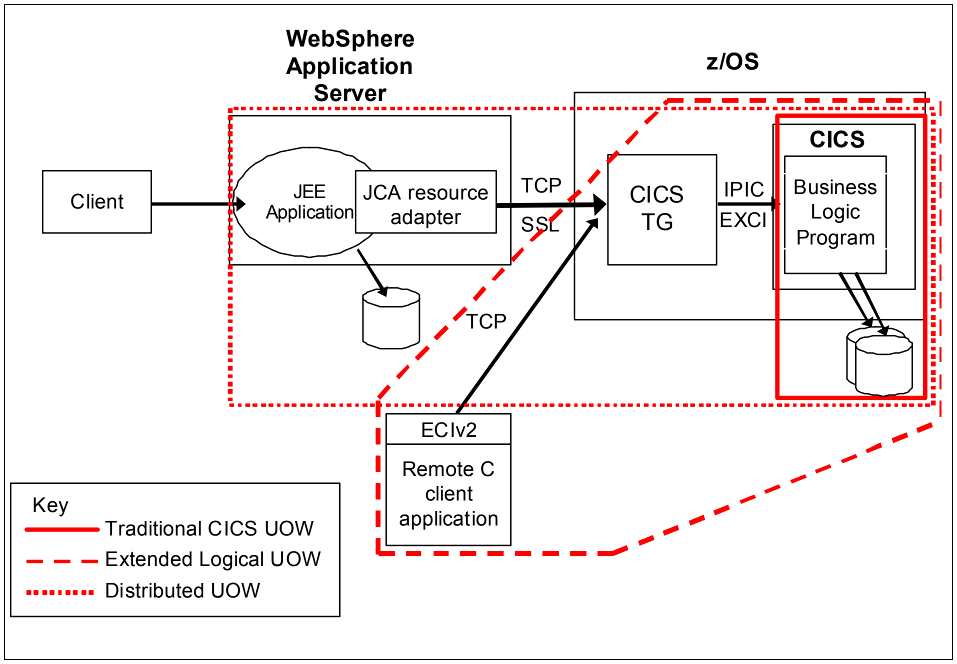


Figure 7-8 Transactional scopes supported by CICS TG

Traditional CICS units of work

For inbound requests into CICS TS, CICS TS will always create a unit of work under which to process recoverable updates, as described in 7.2, “Transactional building blocks” on page 187. This enables updates to multiple CICS resources, spanning multiple CICS regions, to be all committed or all rolled back. When this capability meets your needs, use the ECI Resource Adapter.

This capability is available through JCA and ECI v2.

Extended logical units of work

If all your transactional resource updates are made in CICS TS, you need to make multiple requests to the same CICS region, and you want all the resource updates to be in the same unit of work, you can use extended logical units of work through the ECI Resource Adapter.

This capability is available through JCA and ECI v2.

Distributed units of work

When you need transactional resource updates in CICS TS to be committed/rolled back in the same unit of work as transactional resource updates outside of CICS TS, you can use the ECI XA Resource Adapter.

The use of the ECI XA resource adapter provides the ability for any number of CICS regions to participate in a global transaction with two-phase commit coordination with a number of other XAResource capable resource managers.

This capability is available through JCA, but not through ECI v2.

Optimizations for distributed units of work

Where possible, products such as WebSphere Application Server and CICS TS optimize distributed units of work into a one-phase commit. Examples of these optimizations are *last resource optimization, only agent,* and *read only,* as described in “Optimizations for distributed units of work” on page 192.

These optimizations are available when using JCA from WebSphere Application Server into CICS TS.

Compensating transactions

There is no specific support needed for writing compensating transactions within CICS TS. The user can write an additional CICS program that can be called via the CICS TG to perform the business logic necessary to compensate for a previous transactional update.

Idempotent requests

If the decision is made not to use distributed units of work, then idempotent requests could be used to ensure that in the event of a response message not being received, a request message can be retransmitted without causing the same updates to be made twice at the service provider.

7.5.4 Transactional considerations for CICS TG

The following list of questions and comments will help you to choose between the different tranactional building blocks available for CICS TG:

- Is the traditional CICS unit of work support good enough?

  If all the transactional resources that you need to coordinate are accessed from CICS TS, the traditional CICS unit of work support should meet your requirements. This will be the best-performing option.

- Do extended logical units of work meet your needs?

  If all the transactional resources that you need to coordinate are accessed from CICS TS, but you need to make multiple calls from the invoking application into CICS TS and you want those calls to use the same unit of work, then extended logical units of work could meet your needs. There is no resync capability with extended logical units of work, so if CICS TS is told to commit, does the commit, and then there is a failure, the invoking application will not be able to learn whether the commit occurred.

  With WebSphere Application Server (WAS), you can alternatively use distributed units of work. If CICS TS is the only resource manager making transactional updates then WAS can optimize the two-phase commit to a one-phase commit. This gives the same quality of service as extended logical units of work, but with the benefit that there is a resync capability, so if CICS TS is told to commit, does the commit, and then there is a failure, WebSphere Application Server resyncs with CICS TS when the failure is resolved and can learn the result of the commit.

- Do you need distributed units of work?

  If you need to coordinate updates to resources in CICS TS with updates to resources in other systems synchronously, distributed units of work could be the answer. Alternative solutions include compensating transactions, or using WebSphere MQ with persistent messages if an asynchronous model is appropriate.

- Does my Java EE Application Server support distributed unit of work optimizations?

  If you are using JCA through WebSphere Application Server, the optimizations are supported. If you are using an alternative application server, you will need to refer to your product documentation.

- How will distributed units of work affect high availability?

  Considerations need to include deadlock avoidance and unit of work affinities. Refer to 8.4.3, “High-availability considerations for CICS TG” on page 240, for further details.

- For inbound calls to CICS TS, do your CICS programs already make SYNCPOINT calls?

  If so, you need to change those programs if you want the unit of work to be coordinated from the invoking application. If such a CICS program attempts to issue an explicit SYNCPOINT when the CICS region that it is running in is not coordinating the transaction, `SYNCPOINT` fails with an error return code, which if unhandled causes a program abend. The CICS region can be told to allow syncpoints by invoking the CICS program with SYNCONRETURN, as described in 7.2.1, “Traditional CICS units of work” on page 187, but this means that the CICS updates are not coordinated by the invoking application.

- If you use last resource optimization, will the applications in the distributed unit of work be able to continue to meet the business needs if a failure in the one-phase commit resource results in a mixed outcome to a transaction?

  Applications that exploit last resource optimization are subject to an increased risk of a mixed outcome in a global transaction, if the one-phase commit resource fails during the commit processing. It is important to understand the implications of this to your system.

7.6 WOLA

WebSphere Optimized Local Adapters (WOLA) provide a high-speed bi-directional cross-memory pipe between WebSphere Application Server for z/OS and CICS TS. The support in WebSphere Application Server for z/OS is via JCA. An explanation of JCA can be found in 7.5, “CICS TG for z/OS” on page 202. To enable the WOLA connection through JCA, a JCA resource adapter called ola.rar is provided.

7.6.1 Supported building blocks for WOLA

Figure 7-9 shows different transactional scopes supported by WOLA.

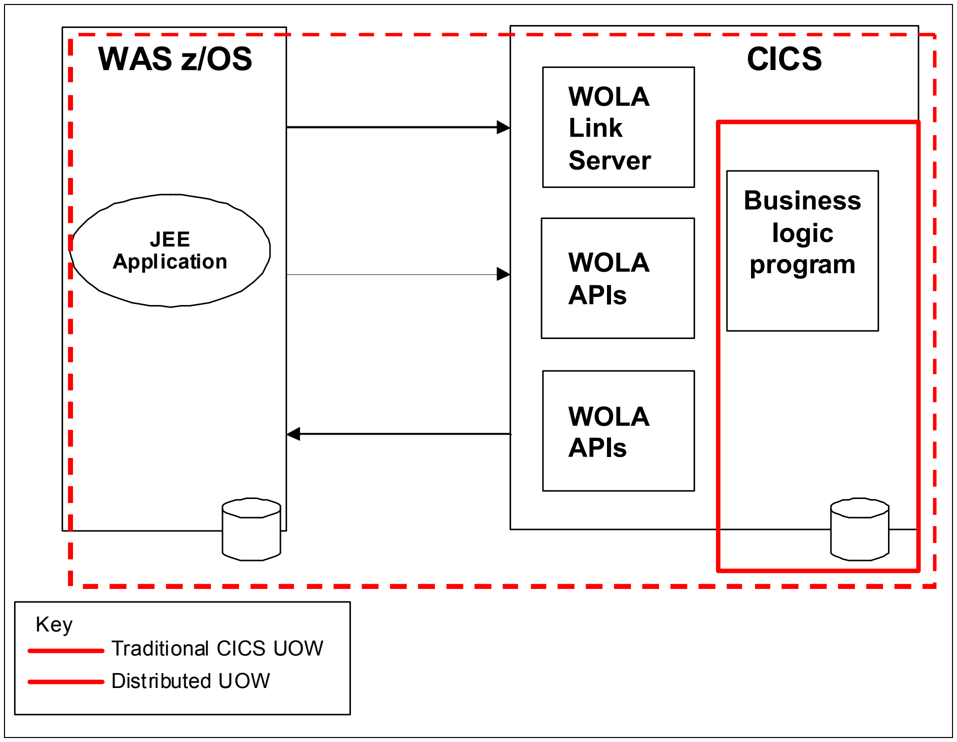


Figure 7-9 Transactional scopes supported by WOLA

Traditional CICS units of work

For inbound requests into CICS TS, CICS TS will always create a unit of work under which to process recoverable updates, as described in 7.2, “Transactional building blocks” on page 187. This enables updates to multiple CICS resources, spanning multiple CICS regions, to be all committed or all rolled back.

Distributed units of work

WOLA provides two-phase commit global transaction support for both CICS to WebSphere Application Server and WebSphere Application Server to CICS program invocations. Both the native language and EJB portions of an application can participate in the same global transaction using two-phase commit when the native language portion of the application is running in CICS TS.

When calling from CICS TS into an optimized local adapter EJB, a z/OS Resource Recovery Services (RRS) unit of recovery token is passed from CICS TS to WebSphere Application Server, which uses this token to create its own unit of recovery that is then cascaded to the unit of recovery received from CICS TS. When the CICS TS transaction reaches a syncpoint, RRS drives the WebSphere Application Server unit of recovery to completion.

When calling from WebSphere Application Server for z/OS into a CICS TS transaction using WOLA, an XA-capable transaction context is passed from WebSphere Application Server to CICS TS. The WOLA CICS link server running in CICS TS reads the XA transaction context and creates a new unit of work to run the specified native language program. When WebSphere Application Server reaches a syncpoint, XA protocol messages are exchanged between WebSphere Application Server and the link server. The link server drives the appropriate CICS functions to complete the unit of work.

To read more about WOLA support for CICS TS, refer to this URL:

`http://www.ibm.com/developerworks/websphere/techjournal/1102_mulvey/1102_mulvey.html?ca=drs-`

Optimizations for distributed units of work

WOLA supports the *only agent* optimization, as described in “Optimizations for distributed units of work” on page 192. This optimization ensures that if CICS TS is the only resource manager participating in a distributed unit of work, the distributed unit of work is optimized to a one-phase commit.

Compensating transactions

There is no specific support needed for writing compensating transactions within CICS TS. The user can write an additional CICS program that can be called via WOLA to perform the business logic necessary to compensate for a previous transactional update.

Idempotent requests

If the decision is made not to use distributed units of work, then idempotent requests can be used to ensure that in the event of a response not being received, a request can be retransmitted without causing the same updates to be made twice at the service provider.

7.6.2 Transactional considerations for WOLA

The following list of questions and comments will help you to choose between the different tranactional building blocks available for WOLA.

- Are you able to run WebSphere Application Server for z/OS and CICS TS on the same LPAR?

  If so, consider WOLA. If not, consider an alternative technology such as CICS TG.

- Is the traditional CICS unit of work support good enough?

  If all the transactional resources that you need to coordinate are accessed from CICS TS, the traditional CICS unit of work support should meet your requirements. This will be the best-performing option.

- Do you need distributed units of work?

  If you need to coordinate updates to resources in CICS TS with updates to resources in other systems synchronously, distributed units of work could be the answer. Alternative solutions include compensating transactions, or using WebSphere MQ if an asynchronous model is appropriate.

- How will distributed units of work affect high availability?

  Considerations need to include deadlock avoidance and unit of work affinities. Refer to 8.5.3, “High-availability considerations for WOLA” on page 243, for further details.

- For inbound calls to CICS TS, do your CICS programs already make SYNCPOINT calls?

  If so, you will need to change those programs if you want the unit of work to be coordinated from the invoking application. If a CICS program attempts to issue an explicit SYNCPOINT when the CICS region that it is running in is not coordinating the transaction, `SYNCPOINT` fail switch an error return code, which if not handled causes a program abend. The CICS region can be told to allow syncpoints by invoking the CICS program with `SYNCONRETURN`, as described in 7.2.1, “Traditional CICS units of work” on page 187, but this means that the CICS updates are not coordinated by the invoking application.

7.7 CICS web support

CICS TS provides built-in support for communications over HTTP. In this section we discuss the transactional scope options for CICS web support.

CICS web support uses standard CICS transactions to process a single request from a web client. Any updates to recoverable resources will be committed or rolled back prior to sending the HTTP response, assuming that an explicit SYNCPOINT request has not been made by the application.

7.7.1 Supported building blocks for CICS web support

Figure 7-10 shows transactional scopes supported by CICS web support.

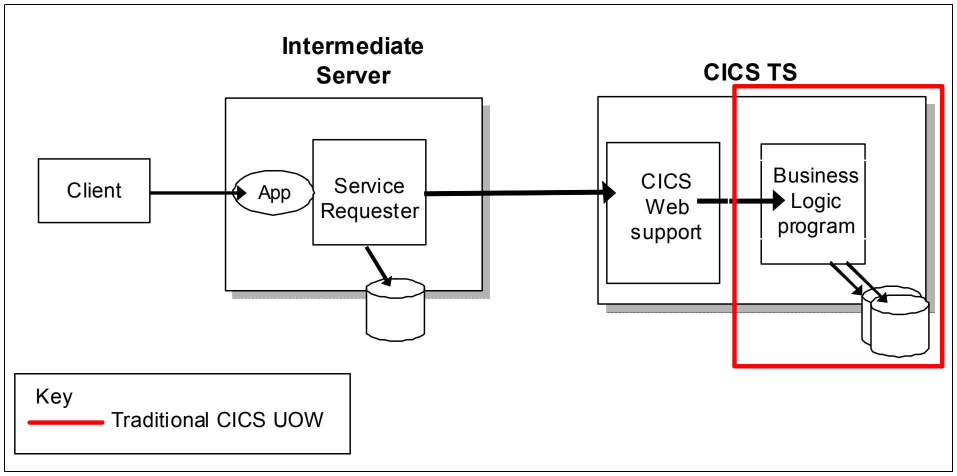


Figure 7-10 Transactional scopes supported by CICS web support

Traditional CICS units of work

For inbound requests into CICS TS, CICS TS will always create a unit of work under which to process recoverable updates, as described in 7.2, “Transactional building blocks” on page 187. This enables updates to multiple CICS resources, spanning multiple CICS regions, to be all committed or all rolled back. This is the default behavior when using CICS web support.

Compensating transactions

There is no specific support needed for writing compensating transactions within CICS TS. The user can write an additional CICS program that can be called via CICS web support to perform the business logic necessary to compensate for a previous transactional update.

Idempotent requests

Idempotent requests could be used to ensure that in the event of a response message not being received, a request message can be retransmitted without causing the same updates to be made twice at the service provider.

7.7.2 Transactional considerations for CICS web support

The following question and comments will help you to choose between the different tranactional building blocks available for CICS web support. Is the traditional CICS unit of work support good enough?

If all the transactional resources that you need to coordinate are accessed from CICS TS, the traditional CICS unit of work support should meet your requirements. If it does not, then you might need to look at an alternative technology that supports a wider range of transactional scopes.

7.8 WebSphere MQ

WebSphere MQ provides an alternative approach to the synchronous request/reply pattern, through its assured delivery and asynchronous messaging qualities of service. Often used as the basis for an enterprise service bus, it provides a common API across multiple platforms and is a popular method for connecting systems together.

7.8.1 Supported building blocks for WebSphere MQ

Figure 7-11 shows transactional scopes supported by CICS TS when used with WebSphere MQ.

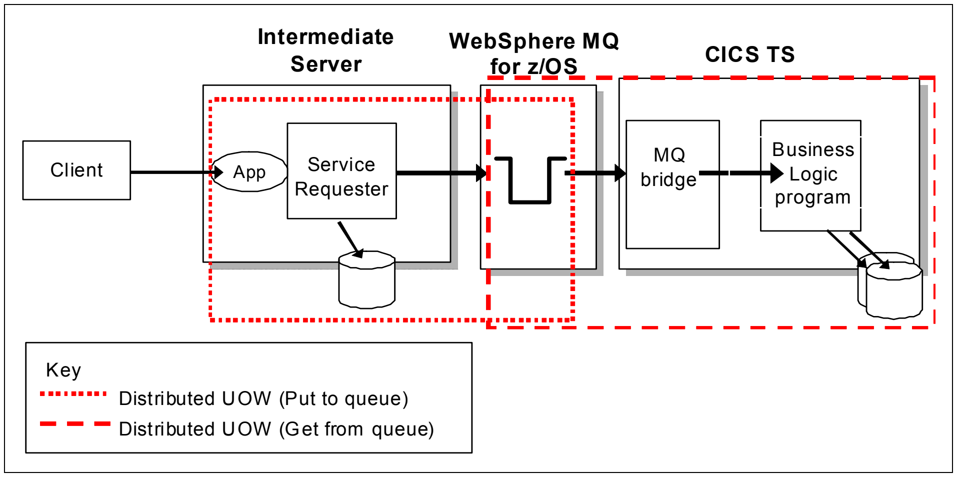


Figure 7-11 Transactional scopes supported by WebSphere MQ

Asynchronous messaging transactional model

Figure 7-11 on page 214 highlights how two different units of work are used in the asynchronous messaging transactional model to perform the asynchronous equivalent of a synchronous distributed unit of work. In the first unit of work, the service requester performs local transactional resource updates and puts a persistent request message to the MQ queue. The unit of work is then committed. In the second unit of work, CICS TS reads the request message from the MQ queue and runs the business logic program that updates transactional resources. The end result is that transactional resources in both the service requester and the service provider are updated as they would be in a distributed unit of work. The difference between the synchronous and asynchronous models is that there is a time delay between the two units of work and there is a reliance on WebSphere MQ to look after the request message until the service provider consumes it.

CICS TS requires a connection to a WebSphere MQ Queue Manager running on the same LPAR as the CICS region. Having got a message from the queue, in the event of an error, any transactional resource updates made by the business logic program are backed out, and the get of the MQ message is also backed out. The message becomes available on the queue again.

It is possible to tell CICS TS that the backend CICS program should be linked to with SYNCONRETURN specified. This is done via a flag set in the request message. When the dynamic program link (DPL) request is made to the CICS program, if that program is remote on another CICS region, the remote CICS region will then take a sync point prior to returning. The effect of this is that the business logic is run in a different unit of work to the get of the request message and the put of the reply message (if there is one).

Extended logical units of work

The WebSphere MQ DPL bridge provides a variation on extended logical units of work. A request message can indicate that it starts a unit of work, continues a unit of work, or ends a unit of work. When the bridge sees a request message that starts a unit of work, it reads that message from the queue outside of the unit of work that it creates to run the target backend CICS programs. It also puts the reply message outside of the unit of work. This means that the reply message is returned immediately to the requesting application, but the unit of work under which the CICS updates were made is kept open. Request messages continue to be read, backend CICS programs are called within the same unit of work, and reply messages are put, until the CICS bridge sees a message that indicates to it that the unit of work should be committed or rolled back.

Compensating transactions

There is no specific support needed for writing compensating transactions within CICS TS. The user can write an additional CICS program that can be called via WebSphere MQ to perform the business logic necessary to compensate for a previous transactional update.

Idempotent requests

If you are implementing pseudo-synchronous requests, where the requester puts a request message and waits for a reply message, then there is the risk that messages could get lost in the event of a failure, or that messages are delayed and the requester times out. Idempotent requests can be used to ensure that in the event of a response message not being received, a request message can be retransmitted without causing the same updates to be made twice at the service provider.

7.8.2 Transactional considerations for WebSphere MQ

The following list of questions and comments will help you to choose between the different tranactional building blocks available for WebSphere MQ:

- Is it okay for transactional updates between the application putting the request message (requester) and the application consuming the request message (provider) to be made asynchronously to each other?

  In an asychronous model, the requester and provider do not need to both be active at the same time. This means that updates made to recoverable resources by the requester will be made before corresponding changes are made at the provider.

- When should you use persistent messages?

  Persistent messages are of most value for one-way asynchronous messaging. If you are using MQ as a pseudo-synchronous request/reply mechanism, then it is likely the requester will time-out waiting for a reply at some point. Therefore, persistent messages are of less value, as the requester will need a method for handling timeouts, such as idempotent requests.

- What if a request message cannot be processed?

  In the asynchronous messaging transactional model described in “Asynchronous messaging transactional model” on page 200, in the event of a failure at the provider, the request message would be rolled back onto the queue and become eligible to be read again. If the failure was caused by the contents of the message, then processing the message a second time could cause the same failure and another backout. This repeated pattern is known as a poison message scenario and is usually handled by stating that if a message is rolled back *n* number of times, it should then be moved to another queue for exception processing. It is the responsibility of the user to provide exception processing logic.

- For inbound calls to CICS TS, do your CICS programs already make SYNCPOINT calls?

  If so, you will need to change those programs if you want the unit of work to be coordinated from the invoking application. If a CICS program attempts to issue an explicit `SYNCPOINT` when the CICS region that it is running in is not coordinating the transaction, `SYNCPOINT` fails with an error return code, which if not handled causes a program abend. The CICS region can be told to allow syncpoints by invoking the CICS program with `SYNCONRETURN`, as described in 7.2.1, "Traditional CICS units of work" on page 187, but this means that the CICS updates are not coordinated by the invoking application.

7.9 CICS sockets

CICS sockets provide a low level of support, and this is reflected in the transactional scope options available.

7.9.1 Supported building blocks for CICS sockets

Figure 7-12 shows transactional scopes supported by CICS sockets.

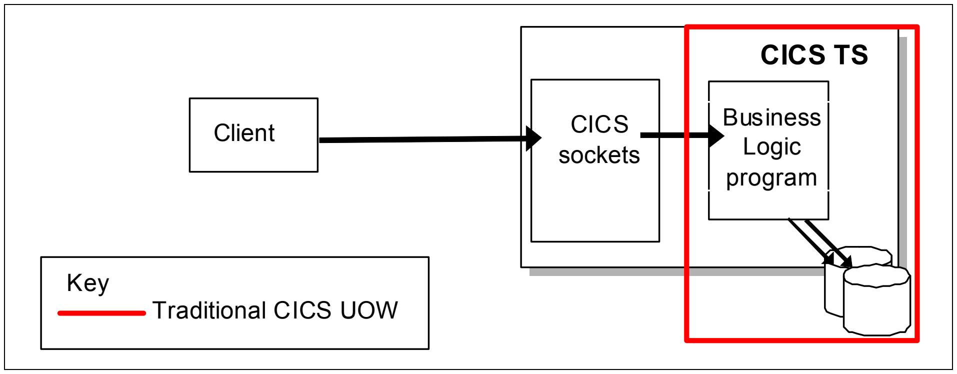


Figure 7-12 Transactional scopes supported by CICS sockets

Traditional CICS units of work

For inbound requests into CICS TS, CICS TS will always create a unit of work under which to process recoverable updates, as described in 7.2, "Transactional building blocks" on page 187. This enables updates to multiple CICS resources, spanning multiple CICS regions, to be all committed or all rolled back. This is the default behavior when using CICS sockets.

Compensating transactions

There is no specific support needed for writing compensating transactions within CICS TS. The user can write an additional CICS program that can be called via CICS sockets to perform the business logic necessary to compensate for a previous transactional update.

Idempotent requests

Idempotent requests could be used to ensure that in the event of a response message not being received, a request message can be retransmitted without causing the same updates to be made twice at the service provider.

7.9.2 Transactional considerations for CICS sockets

The following question and comments will help you to choose between the different tranactional building blocks available for CICS sockets. Is the traditional CICS unit of work support good enough?

If all the transactional resources that you need to coordinate are accessed from CICS TS, the traditional CICS unit of work support should meet your requirements. If it does not, then you might need to look at an alternative technology that supports a wider range of transactional scopes.

8

High availability and scalability

Continuous availability is a key quality of the z/OS operating environment. In this chapter, we start by providing an introduction to high availability and the technologies applicable to the z/OS operating environment. Then for each of the CICS access technologies, we discuss the best approach for optimizing the availability of the solution, and the resulting implications on the scalability of the system.

These are the key questions that we consider for each solution:

- What are the benefits of a high-availability configuration?
- What are the barriers to adopting high availability?
- How is each technology likely to scale?
- How do you balance IP network connections?
- How do you route requests between CICS regions?

8.1 High-availability objectives

For business-critical applications to be available at all times, the underlying IT infrastructure must be able to support continuous service availability across planned and unplanned outages and be able to accommodate varying usage levels without failure. A well-designed high-availability infrastructure can provide a solution to these issues by building on the unique technology provided in a System z Parallel Sysplex®.

Benefits of a highly available infrastructure

The cost of downtime is so great that many of today's enterprises can no longer afford planned or unplanned outages. Even beyond the financial aspects, downtime can also affect key areas of customer loyalty, market competitiveness, and regulatory compliance. A high-availability infrastructure provides a solution to these issues by allowing the system as a whole to continue to function even if individual components cease to function.

Barriers to implementation

Although high-availability configurations are highly desirable, implementation can be hampered by a variety of technical factors. The following list summarizes some of the key factors that might be of consideration:

- Management of state

  For CICS applications this is often caused by application affinities due to information stored in CICS temporary storage or transient data queues that are local to the CICS region. It might also be due to the usage of terminal or session identifiers that are unique to a transaction instance.

- Persistent connections

  Network-based workload balancing techniques such as port sharing or Sysplex Distributor work at the IP socket level, and so are only effective if connections are periodically established, allowing the connection balancing component to distribute requests among the available servers.

- Transactional recovery implications

  For CICS applications this is a specific concern when dealing with distributed units-of-work where multiple updates to resource managers need to be coordinated. For further details on the different transaction models supported in CICS refer to Chapter 7, “Transactional scope” on page 185.

- Additional complexity

  Using additional software or hardware components in a high-availability infrastructure might be prohibitive. Examples are the additional overhead of routing via the z/OS coupling facility, or the additional pathlength of routing via additional CICS systems or of using shared resource manages such as VSAM RLS.

- Fail-back

  If components (such as CICS regions) are restarted, it might not be possible for work to be automatically switched back to the restarted systems.

- Uneven distribution

  Unbalanced distribution across software components can cause problems, such as over usage of CPU in specific CICS regions or z/OS LPARs.

Problems that an HA solution might not solve

A highly available configuration can address many issues, but it does not necessarily solve all potential availability issues. The following list summarizes the key problems be considered when designing an HA infrastructure:

- Storm drain

  All requests are sent to a failing system, as this is seen by the routing component as the optimum location due to a low number of connections or the rapid response time of failing requests. This can be mitigated through automated operations to terminate failing systems, or through the use of the WLM server-specific recommendations.

- Killer applications

  An application that has adverse affects on the target system to which it is sent, causing each system in turn to fail as it is distributed across all available destinations. Isolating applications into different towers can help address this issue.

In the following sections we provide an overview of the z/OS solutions for balancing IP connections and dynamically routing CICS requests between CICS regions. Further details about how they can be exploited by specific CICS integrating technologies are supplied later in this chapter.

8.1.1 IP connection balancing

IP connection balancing is a widely adopted mechanism for distributing socket connections across clusters of servers. There are many implementations provided both in specialist hardware and in routing software. On z/OS the port sharing and Sysplex Distributor functions of Communications Server provide highly efficient and customisable solutions that operate within the heart of the Parallel Sysplex.

Port sharing

TCP/IP port sharing is a unique feature provided by the z/OS Communications Server, which enables a group of cloned servers to listen on the same port. New connections are distributed across the available servers using a weighted round-robin algorithm based on the efficiency of the server application in accepting new connection requests and managing the socket backlog queue. Additionally, feedback from WLM can also be used to influence distribution (see "WLM server-specific recommendations" on page 224).

Port sharing is a very simple feature to implement that can rapidly detect a failed server and that can be exploited by any z/OS IP-based application, including CICS regions, WebSphere MQ queue managers, and the CICS Transaction Gateway.

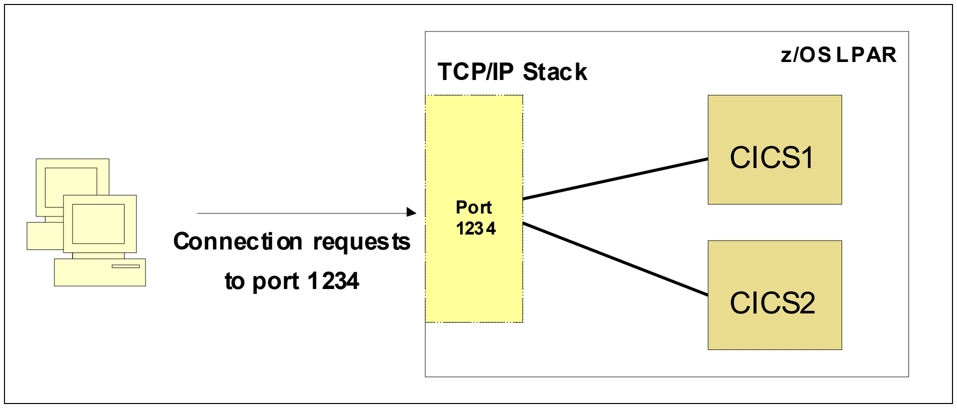


Figure 8-1 Port sharing

For recovery reasons, IP connection balancing technology, such as port sharing or Sysplex Distributor, cannot be used to dynamically balance IPIC-based connections into CICS regions.

Sysplex Distributor

Sysplex Distributor offers the ability to distribute IP connections across cloned servers running on different IP stacks (usually on different LPARs). It is based on the virtual IP address (VIPA) technology in the z/OS Communications Server and provides three key high availability functions:

- Distribution of connections across a sysplex cluster

  This provides the ability to implement a dynamic VIPA as a single network-visible IP address for a set of servers that belong to the same cluster. A client located anywhere in the IP network will see the cluster as one IP address, regardless of the number of server instances that it actually includes. Individual connection requests will be distributed across the server instances according to a variety of criteria such as responsiveness of the server in accepting connections and feedback from MVS workload manager.

- Dynamic VIPA activation

  Dynamic VIPA activation allows an application to create and activate an IP address so that the IP address moves when the application moves between LPARs in the sysplex.

- VIPA takeover and takeback

  Another IP stack in the sysplex can be configured as the backup for a continuously active dynamic VIPA (DVIPA), such that the DVIPA is automatically activated on an alternate stack whenever the primary IP stack suffers an outage. In addition, the primary stack can *take back* the DVIPA after the failed stack has been restored. This takeback is non-disruptive to existing connections, and the takeback is not delayed if existing socket connections are open. This provides redundancy at the IP stack level with minimal impact to the functioning of the server applications.

As such, Sysplex Distributor offers an efficient means of distributing IP requests from within the heart of a z/OS Parallel Sysplex and can be used in conjunction with port sharing to create a sysplex-wide high-availability cluster of cloned servers. Figure 8-2 shows a scenario showing port sharing and Sysplex Distributor use by CICS regions.

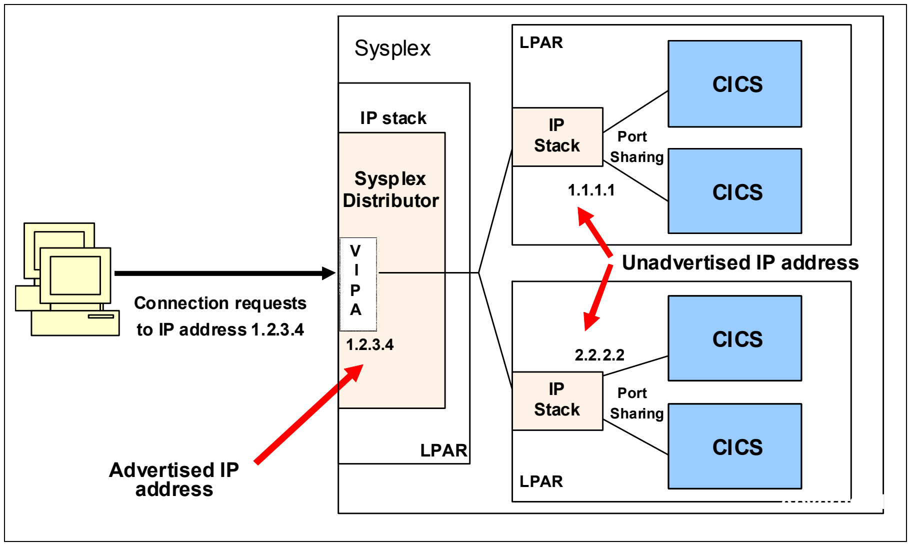


Figure 8-2 Sysplex distributor and port sharing

A more detailed discussion of Sysplex Distributor can be found in *IBM z/OS V1R12 Communications Server TCP/IP Implementation: Volume 3 High Availability, Scalability, and Performance*, SG24-7898.

WLM server-specific recommendations

MVS workload manager (WLM) can optionally be exploited by both port sharing and Sysplex Distributor when making routing decisions for TCP/IP socket connections. The feedback takes into account a variety of WLM factors, including service classes, velocity goals, displaceable CPU capacity, and optionally abnormal transaction completion and server-specific health. For further details about its potential usage with CICS integration technologies, see 8.4, “CICS TG for z/OS” on page 235, and 8.6, “CICS web support” on page 243.

8.1.2 CICSPlex SM workload manager

Typically, CICS regions are used in a cluster termed a CICSplex, which can then be managed by the CICSPlex Systems Manager component to create a managed cluster termed a *CICSPlex* (Figure 8-3). This provides increased capacity beyond that of a single CICS region and also enables different sub-groups of regions to be used for specialized purposes.

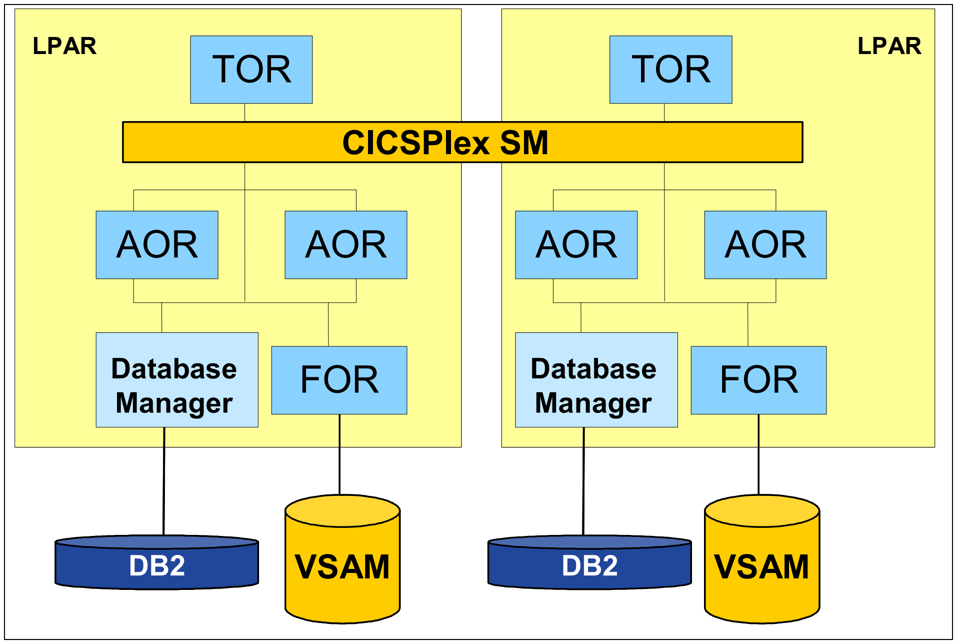


Figure 8-3 CICS dynamic routing

CICS regions that accept incoming requests are designated as terminal owning regions (TORs) or listener regions and are responsible for receiving requests from clients and routing them onto specialized application-owning regions

(AORs). Further specialized file owning regions (FORs) can also be used to provide shared file access to VSAM or other file stores.

CICSPlex SM workload manager provides three key functions for dynamic distribution of requests between CICS regions:

- Workload distribution

  For a given unit of work, CICSPlex SM selects the target region expected to provide the best response time, taking into account the health and capacity of the target regions.

- Workload affinity

  CICSPlex SM honors requirements to route successive transactions to the same target until a defined event causes the affinity to end.

- Workload separation

  CICSPlex SM routes transactions within target scopes based on specific criteria such as transaction ID, user ID, or terminal name.

Dynamic routing models

CICSPlex SM provides two principal routing algorithms for the distribution of work across target CICS regions:

- Queue algorithm

  This calculates a weight based on the existing transaction load in the target CICS region together with the probability of the transaction abending in the target region.

- Goal algorithm

  This calculates a weight based on the ability of target regions to meet response time goals defined in z/OS workload manager for each transaction's service class.

For both of these algorithms link weights are also factored in so that target regions are favored if they are local to the same LPAR or use a more efficient connection type. This provides a bias in favor of sending work over faster connections, expecting better response times as a result.

CICS TS V4.2: Additionally, in CICS TS v4.2 link neutral versions of these routing algorithms can be selected, whereby the locality of the target region does not factor in the routing decision. This removes the bias in favor of sending work to local regions, providing a more even distribution of requests across multiple LPARs in the CICSPlex.

Avoiding transactional deadlocks

A key consideration when creating a CICS dynamic routing infrastructure is the avoidance of transactional deadlocks. This can occur if two requests within the same distributed unit-of-work (UOW) are dynamically routed to two different AORs, but subsequently access the same recoverable resource (such as a shared VSAM file). If this occurs the second request in the UOW will suspend, as it will wait for a lock when trying to update the shared resource, which has already been locked by the first CICS task (Figure 8-4).

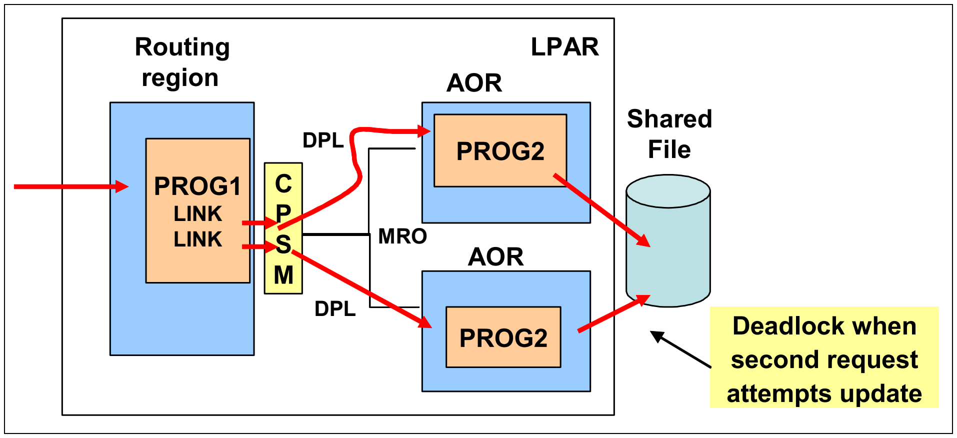


Figure 8-4 Transactional deadlocks

CICS TS V4.2: To avoid transactional deadlocks it is often necessary to ensure that all DPL requests that are dynamically routed to remote regions are routed to the same AOR if they are within the same distributed unit-of-work. CICS TS V4.2 provides built-in support for this through CICSPlex SM WLM support for UOW affinities.

8.2 Scaling

CICS has proven to be a high-performing and scalable transaction-processing system. Good performance implies the efficient use of computing resources and a resulting rapid response time. Scalability implies that the system is capable of running a large number of simultaneous transactions without a significant degradation in response time and that the cost of running a transaction does not increase as the workload increases. A truly scalable solution can therefore be regarded as one that initially performs well at a low rate of usage and whose throughput increases in linear proportion to the number of end users, up to the point at which it saturates the system. This is known as *positive linear scaling*.

The two graphs in Figure 8-5 illustrate how a linearly scalable system should behave. The plot of CPU usage against throughput (transactions per second) shows that the cost per transaction should remain constant as the workload increases. The plot of throughput against the number of users shows a linear increase as workload increases, until a resource constraint is reached. The plots are simplistic and are for illustration purposes only. The assumption is made that each user is initiating the same amount of work.

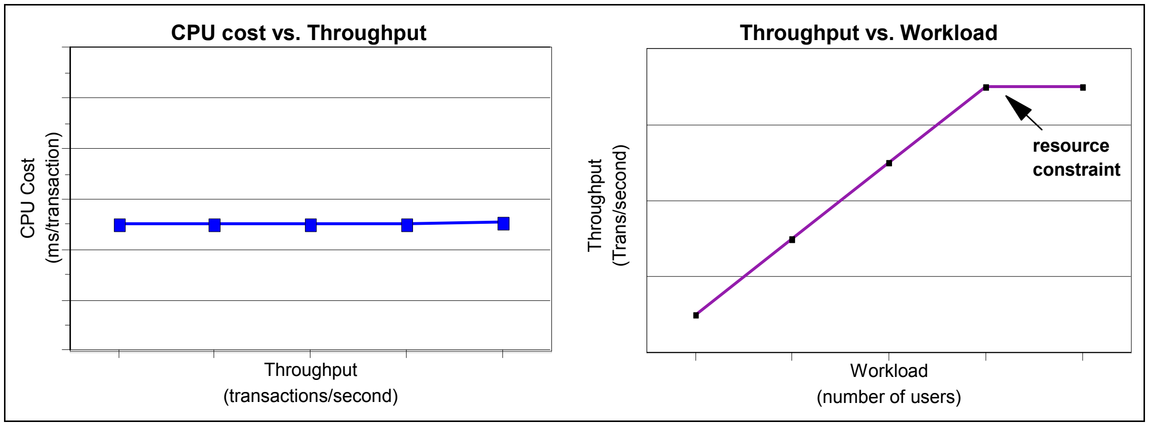


Figure 8-5 Performance of a linearly scalable system

Because computer systems are not infinitely fast, there is always a limit at which the system is saturated. The question is whether this limit is acceptable, and what is the limiting factor.

Limiting factors

CICS provides a wide range of facilities to help applications scale when faced with increasing system load. However, there are key considerations that need to be taken into account to ensure that applications can scale to their maximum extent:

- Multitasking

  A single CICS region has the potential to run up to 999 simultaneous tasks. Each instance of a threadsafe application is allocated to a TCB and runs in parallel with other applications to fully utilize available processors. However, all non-threadsafe applications are dispatched to run on a single quasi-reentrant (QR) TCB when it becomes available, and are therefore limited in scalability by the capacity of a single CPU and the frequency at which other applications give up control. If your applications are non-threadsafe then the best way to scale beyond the bounds of a single

CPU is to dispatch work across multiple CICS regions using dynamic routing, which can also provide redundancy, in addition to increased scaling.

- TCB switching

  Depending on how CICS programs are defined, what resource managers they invoke (such as DB2 or WebSphere MQ), and which CICS commands they execute, CICS tasks can either be dispatched on the QR TCB or on an open TCB (such as an L8, L9, or T8). Each TCB switch has a considerable overhead (around 2,000 instructions), and application design needs to minimize these switches while balancing the need to dispatch work across multiple CPUs. For a more detailed discussion about the implication of TDB switching in CICS, see *Threadsafe Considerations for CICS*, SG24-6351.

- Optimizers and appliances

  CICS provides the ability to use a variety of optimizers and appliances to improve the scalability of applications. These include the following range of options:

  - System z Application Assist Processor (zAAP)

    zAAP processors are speciality processors that can be used to offload Java workloads from the System z general-purpose CPs, with the aim of reducing processing costs. Any applications that run within a CICS JVM server or JVM Pool can be offloaded to zAAP, which includes Java applications and PHP or Groovy applications, in addition to SOAP processing that runs using the Axis2 pipeline.

  - System z Integrated Information Processors (zIIPs)

    zIIP processors are similar in concept to zAAPs but are designed to offload TCP/IP and XML processing. When used with CICS, they can be used to offload TCP/IP messages over 32 KB when using Hipersocket connections between LPARs and can assist with XML parsing through support for the XML system services parser.

    Tip: z/OS V1.11 added a new capability that enables zAAP-eligible workloads to run on the zIIP. This new capability is ideal for customers without enough zAAP or zIIP eligible workload to justify a specific speciality engine.

  - Cryptographic coprocessors, such as the CPACF and CEX2, can be used to accelerate public and private key encryption. For further details see Chapter 6, “Security” on page 133.

  - WebSphere DataPower appliance: WebSphere DataPower is a purpose-built SOA appliance for delivering optimized and scalable SOA solutions. As specialized SOA hardware, it can be used to offload expensive operations from middleware solutions such as CICS, thus improving the overall scalability of the solution. Typical operations include the processing of large XML messages, validation of XML digital signatures, and XML schema-validation. These DataPower capabilities are also available in the DataPower XI50z, which is a blade-form factor that is installed in the zEnterprise Blade Extension (zBX).

8.3 CICS Web services

Web services provides an open standards based communications method based on XML-formatted SOAP messages and enable interoperability between a wide range of service requesters and providers. CICS can act as both a service requester and provider, and there are different considerations when using these two scenarios. In the following section we analyze the different choices available at the IP and CICS routing layers when creating a highly available infrastructure, and provide recommendations in terms of best practices for the different choices available.

8.3.1 Creating an HA infrastructure

There are two underlying transports supported by the CICS Web services support:

- SOAP/HTTP
- SOAP/ MQ

Both of these transports support the creation of high-availability connections using either IP connection balancing or WebSphere MQ load balancing techniques. For further details about how each of these technologies can be used with CICS Web services, refer to 8.6, “CICS web support” on page 243 and 8.7, “WebSphere MQ” on page 249.

Dynamic routing

CICS Web service provider applications support two dynamic routing models for the routing of requests between listening regions and AORs:

- DPL
- Pipeline transaction routing

DPL routing

In this scenario (Figure 8-6) the routing occurs when the pipeline task links to the target business logic program after the inbound SOAP request has been mapped to a COMMAREA or a container. The DPL routing is controlled by the program specified in the CICS DTRPGM system initialization parameter. These are the advantages of this scenario:

- DPL routing supports MRO, IPIC, or APPC connections.
- DPL routing can be used with both CICS and Axis2 pipelines.
- DPL routing generally is more efficient, as the SOAP message has already been parsed and transformed into a COMMAREA or containers.
- Setup is much simpler to perform, as no pipeline resources are required in the AOR.

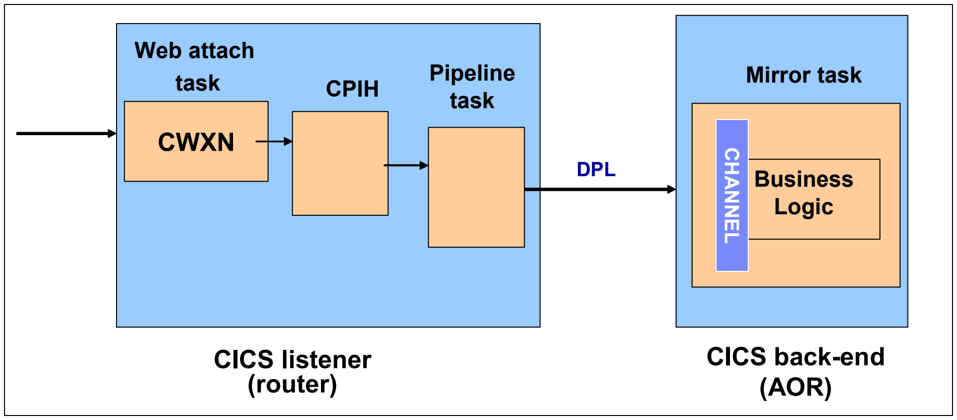


Figure 8-6 DPL routing

However, due to the way that CICS dynamic DPL functions, the transaction ID used in the AOR to execute the DPL request will default to either the value specified in the local program definition or the default mirror transaction (CSMI). For monitoring reasons it might be beneficial to make this the same as the transaction ID used for the pipeline task, which can be achieved using the XPCREQ global user exit.

Pipeline transaction routing

In this scenario (Figure 8-7) the entire pipeline task is routed from the listening region to the AOR, where further processing of the SOAP message occurs before the program links to the business logic. The transaction that runs the pipeline and business processing is eligible for routing when the transaction is defined with DYNAMIC=YES. The routing is controlled by the program specified in the DSRTPGM system initialization parameter. These are key considerations of this model:

- Pipeline routing is only supported for MRO connections, so it is limited to CICS regions within the sysplex.
- Pipeline routing is not supported for Axis2 web service Java applications (where the complete processing of the web service take place with the CICS JVM).
- The link to the business logic in the AOR runs under the pipeline alias task, which can be useful for monitoring and accounting.

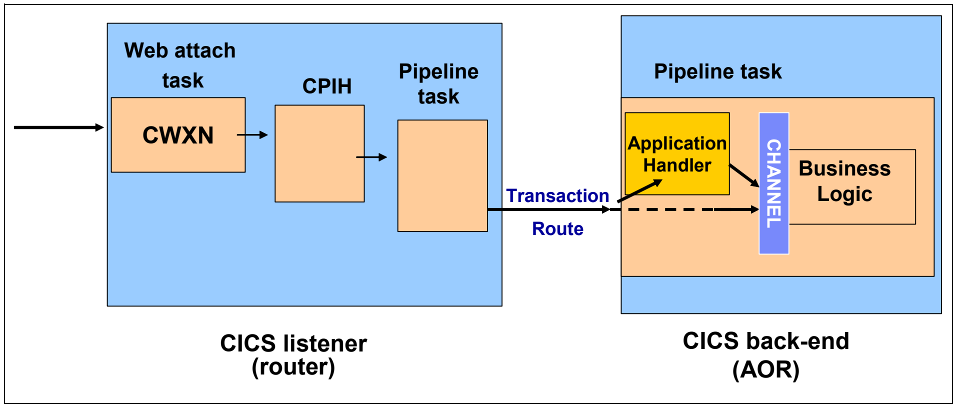


Figure 8-7 Pipeline routing

For an example on using inbound and outbound Gateway-owning regions in a CICS Web services high-availability configuration, see Chapter 10, "CICS TG for z/OS scenario" on page 275. For further details about CICS workload management, refer to the IBM Redbooks publication *CICS Web Services Workload Management and Availability*, SG24-7144.

Using two-phase commit

Web Services Atomic Transactions (WS-AT) can be used to provide two-phase commit support for distributed transactions using web services in CICS. WS-AT is documented in detail in Chapter 7, "Transactional scope" on page 185. When WS-AT is used with a CICS Web service provider application special considerations have to be taken to ensure that transactional recovery can be guaranteed to complete between the web service requester and provider. CICS provides a solution to this in the form of the WS-AT directory data set (DFHPIDIR). This data set is designed to be shared between CICS regions involved in pipeline processing for WS-AT. All the regions that use the data set must be connected by MRO connections to allow dynamic routing of the pipeline to the correct region (Figure 8-8). Once shared it enables creation of a single logical web service provider consisting of multiple instances of CICS regions acting as web service providers, listening on the same port and IP address (when using SOAP/HTTP) or the same shared queue (when using SOAP/MQ).

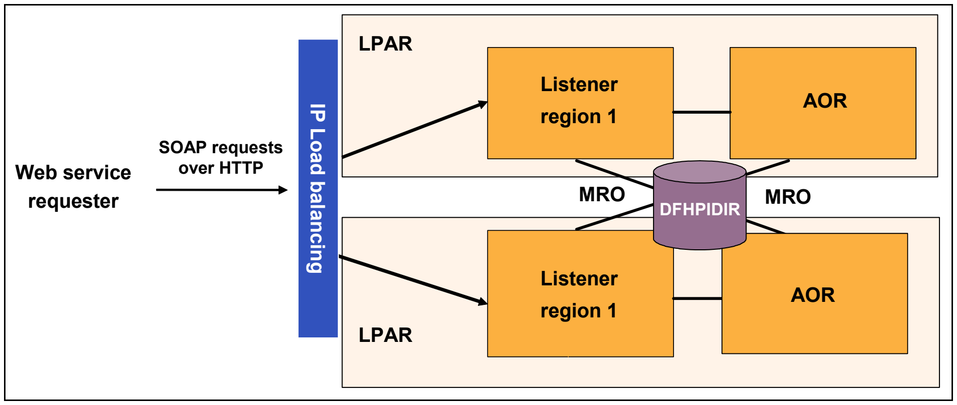


Figure 8-8 IP connection balancing with Web Service Atomic Transactions

For further details about designing and configuring WS-AT configurations in CICS refer to *Implementing CICS Web Services*, SG24-7657.

8.3.2 Scaling

The scalability of CICS Web service applications is primarily dependant on the size and complexity of the SOAP messages. The CPU consumed by the parsing process has been well studied, and these are the general rules to consider when creating a web service:

- Keep message size small and message structure simple.
- If large amounts of data need to be exchanged, consider the use of binary optimization using MTOM. For more details see Chapter 5, “Application interfaces” on page 89.
- If you cannot use options 1 or 2 within your web service provider application consider an optimizer technology such as DataPower, which can perform the specialized parsing function, or usage of the Axis2 pipeline, which can offload XML parsing to zAAPs speciality processors. For further details about how to

calculate the CPU consumed by the CICS parsing process refer to *A Guide to CICS Web Services Performance,* SG247354.

Using DataPower to scale

WebSphere DataPower appliances provide a range of abilities to offload key components of SOAP message processing from the CICS region. If the message is very long or complex, then CPU cost is proportional to the length/complexity, and in this case it might be worth the additional development effort to manage the body XML processing in DataPower. In this case there are basically two ways for DataPower to transfer the XML-parsed binary data for usage by CICS:

- Wrapped in a MTOM attachment and using a CICS MTOM Web service
- Using WebSphere MQ

The choice for a particular scenario depends on a variety of factors, such as existing usage of WebSphere MQ, and if there is a requirement to reuse the same access mechanisms for all services.

Axis2

In CICS TS V4.2 there is an additional option of running the Axis2 web services framework inside a CICS JVM server. This supports a variety of configurations, as outlined in Chapter 3, “Technology overview” on page 37. Usage of Axis2 allows for maximum zAAP offload and is likely to reduce general CPU usage in either of these cases:

- If the XML messages are large and complex
- If the web service provider application is developed using a native Java SOAP processing API such as JAX-WS

Note: Although general CPU usage might be reduced with the Axis2 SOAP pipeline, and this can improve scalability if CPU is limited, it might have a detrimental affect on response times, as overall CPU usage is likely to increase.

8.3.3 High-availability considerations for CICS Web services

When designing your CICS integration solution with CICS Web services, the following questions are the key considerations that will need to make:

- Which transport should you use (WebSphere MQ versus HTTP)?

  WebSphere MQ and HTTP transports are both highly scalable, and the CPU cost of the two transports is broadly similar. However, the total CPU transport cost of SOAP/HTTP is around 10% lower than SOAP/MQ, although the usage of message queuing can provide the benefit of assured delivery and better resiliency in cases of temporary network failures.

  Both transports offer connection balancing techniques, however, message queuing offers a wider variety of connection distribution options, both at the IP level and at the message level.

- Which CICS routing model should be used?

  CICS dynamic routing offers both pipeline transaction routing and DPL as choices for routing of web services requests. DPL is invariably the best choice, as it is simpler to configure, supports all topologies, and generally performs better as the requests routed to the AORs are not XML formatted.

- When do you use DataPower?

  The CPU cost of parsing web service messages is proportional to the length and complexity of the XML message. If the message is long or complex then it is often worth the additional development effort to manage the XML body processing in DataPower and transform this into a binary payload using WebSphere MQ or MTOM. Additionally, if WS-security digital signature processing functions are required, then the overhead of processing the digital signature information can be offloaded to WebSphere DataPower (see “z/OS identity propagation support with CICS TG” on page 163).

- What is the impact of two-phase commit?

  WS-AT can be used to provide two-phase commit with the CICS or Axis2 pipeline. However, as with any use of distributed transactions, the number of interactions during the scope of the unit-of-work can have significant affects on scalability, as CICS transactions will remain active for longer and locks will be held for longer on recoverable resources. In addition, use of WS-AT with cloned listener regions requires additional setup using a shared WS-AT directory data to ensure that transactional integrity is preserved.

8.4 CICS TG for z/OS

The CICS Transaction Gateway for z/OS provides a connector infrastructure for quickly providing access to existing CICS applications. This includes a mature set of functions that can be integrated with other software components across the Parallel Sysplex to provide a fully redundant and highly available connector technology.

8.4.1 Creating an HA infrastructure

There are two key issues when creating a highly available CICS TG infrastructure:

- Gateway groups
- CICS server selection

Gateway cloning and IP connection balancing

The Gateway daemon is the runtime component of the CICS TG that handles incoming client connections and routes work to the appropriate CICS system. To provide for redundancy, a highly available Gateway group can be created consisting of a cluster of cloned Gateway daemons. The Gateway daemons must all use the same configuration and must be listening on the same external IP endpoint, which can be achieved by using port sharing within the same LPAR or Sysplex Distributor across LPARs within a sysplex (Figure 8-9).

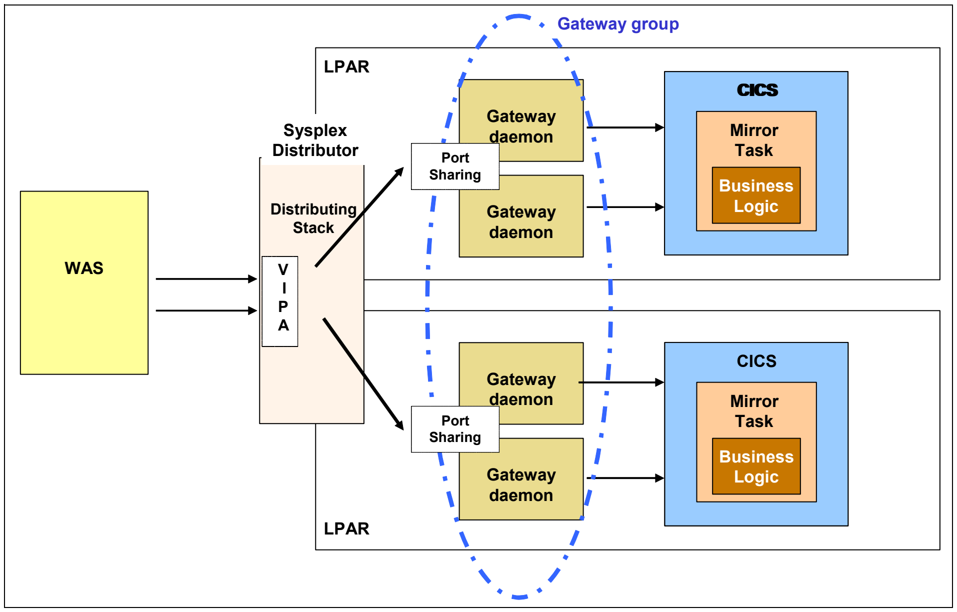


Figure 8-9 Gateway groups and IP connection balancing

Each Gateway daemon in the Gateway group works in conjunction with RRS and is capable of providing peer recovery with the connected CICS regions in case of in-doubt failures during two-phase commit XA transaction processing.

Dynamic server selection

Dynamic server selection (DSS) is a function of the CICS TG on z/OS that provides the ability for the CICS TG run time to dynamically select a CICS server for any given ECI request (Figure 8-10 on page 237). This feature allows each Gateway daemon in a Gateway group to route requests appropriately, for instance, selecting a local CICS region or routing specific requests to a specialized group of AORs. This function is also integrated with the two-phase commit XA transaction support provided by the CICS TG and is designed to ensure that after a CICS server has been dynamically selected for any given ECI request, all future requests within the scope of the unit-of-work will be routed to the same CICS server until the unit-of-work completes.

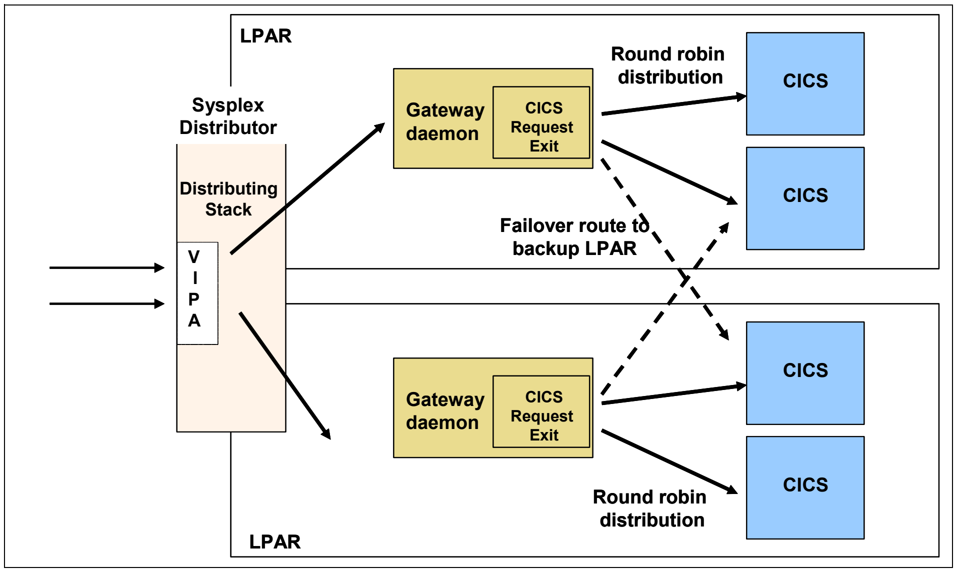


Figure 8-10 Gateway groups and dynamic server selection

In CICS TG for z/OS V8.1, DSS can either be defined in policies within the CICS TG configuration or it can be controlled dynamically using the CICS request exit. Policy-based DSS supports generic or server-specific rules and provides a choice of round-robin or failover algorithms.

The CICS request exit is a user exit deployed into a Gateway daemon. A fully functional working sample that uses text-based configuration files is provided in CICS SupportPac CA1T. This SupportPac provides the ability to configure workload management via round-robin or failover polices, with primary and secondary levels of failover and configureable time-outs to enable fail back to primary CICS regions after they are restarted and online policy updates.

Note: The CICS EXCI user replaceable module, DFHXCURM, also provides the ability to retry failed ECI requests. However, its usage is no longer recommended with the CICS TG, as it does not support IPIC connections and is not integrated with CICS TG statistics and systems monitoring.

WLM and server-specific health

The CICS TG for z/OS can be configured to report server-specific health values to WLM. These health values are dynamically calculated for a given Gateway daemon as a percentage value based on the number of failing ECI requests in a configured interval. The health values for each Gateway daemon will then be used by IBM Communications Server during the distribution of new socket connections by port sharing or Sysplex Distributor. To enable this support it is necessary to both enable health reporting support in each Gateway daemon and to enable WLM weights to be used for a shared port or DVIPA.

Server-specific health can be used in conjunction with DSS to provide a mechanism to prevent *storm drain* scenarios by the removal of a specific Gateway daemon from IP connection balancing within the sysplex when all of its available CICS servers are offline (Figure 8-11). In addition, because WLM also provides feedback on displaceable CPU capacity, it can be useful in allocating connections to LPARs where there is sufficient general-purpose or zAAP processor to run the required workload.

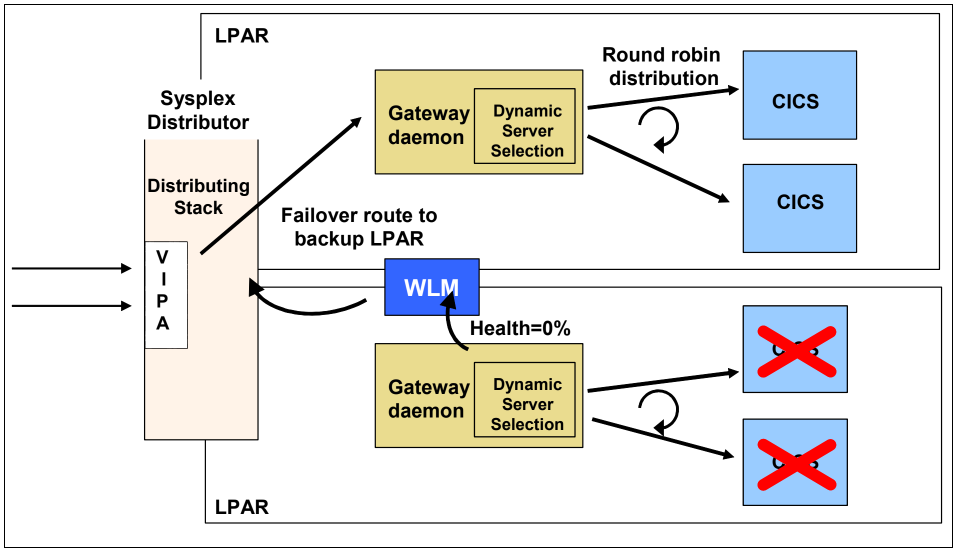


Figure 8-11 CICS TG and health reporting with port sharing

8.4.2 Scaling

The CICS TG for z/OS provides a highly optimized inbound CICS connector infrastructure for a wide variety of clients. The key factor affecting the scalability of a CICS TG solution is usually the payload size of each request. The amount of data transmitted for each payload depends on both the application design and whether COMMAREA or channel-based requests are being sent to CICS:

- When using channel-based requests, only containers within the channel that have been modified are returned back to the calling application. Therefore, an efficient design ensures that different containers are used for input and output to ensure that only the data required is transmitted.
- When using COMMAREA-based requests, CICS provides the ability to dynamically truncate the inbound or outbound data flows that contain trailing blank (null) data.
- XA two-phase commit provides an optimized mechanism for supporting distributed units of work between JEE application servers and CICS, with a small delta cost per transaction. However, as with any use of distributed transactions, the number of interactions during the scope of the unit-of-work can have significant affects on scalability, as CICS transactions will remain active for longer and locks will be held for longer on recoverable resources. See Chapter 7, "Transactional scope" on page 185, for more information.
- Using SSL for security can add overhead both to the process of connection establishment and also to the encryption of each request. The first priority is to ensure that connections are re-used wherever possible, as the most significant overhead is the SSL handshake during connection establishment. Additionally, usage of SSL cipher suites, such as DES, AES, or Triple DES, that are supported using the System z CPACF hardware ensures that the cost of encryption is a low as possible.

For further details on managing the CICS TG refer to *Exploring Systems Monitoring for CICS Transaction Gateway V7.1 for z/OS*, SG24-7562.

8.4.3 High-availability considerations for CICS TG

This section summarizes the key decision factors affecting a high-availability configuration with the CICS TG for z/OS.

When to use Dynamic Server selection or WLM health

Use Dynamic Server selection or WLM health in these situations:

- Policy-based DSS supports CICS region failover and failback with zero failures and is easy to configured and deploy.
- Using the CICS Request Exit gives a highly customizable solution for primary distribution to local CICS regions and secondary failover to CICS regions on remote LPARs, with the ability to change definitions at run time.
- WLM can work in conjunction with DSS to provide the ability to remove a Gateway from IP connection balancing if all available CICS servers are unavailable. However, it requires additional operator intervention or automation to perform fail-back to a given Gateway daemon after CICS regions are restarted after failure. However, it can also be used in a multi-LPAR scenario to help distribute requests to systems with more available CPU.

How to handle failback for restarted Gateway daemons

If IP connection balancing is in use across a group of cloned Gateway daemons, then restarted Gateways might not be allocated connections if all existing connections are being reused from a connection pool. To ensure that such Gateway daemons can be efficiently used in a failback situation, we suggest that the CICS TG *idletimeout* or the WebSphere Pool property for *aged* or *unused* time-out are set to ensure that connections are periodically recycled.

Impact of XA global transactions on a HA solution

These are details of the impact of XA global transations on an HA solution:

- Using XA global transactions requires the creation of a highly available Gateway group with identical configurations and access to RRS.
- If CICSPlex SM is already in use for routing DPL requests from routing regions to AORs, then CICS TS V4.2 UOW affinity support should be used to ensure that all requests within an XA global transaction are routed to the same AOR to prevent potential transactional deadlocks. For more details see “Avoiding transactional deadlocks” on page 226.
- If ECI requests are to be routed from a Gateway daemon to a CICS region on a different LPAR, then IPIC connections must be used, as EXCI does not support the use of RRS for transactional requests sent between LPARs.

8.5 WOLA

WebSphere Optimized Local Adapters (WOLA) is a functional component of WebSphere Application Server for z/OS that provides an efficient cross-memory mechanism for calls from both inbound to WAS z/OS and outbound from WAS z/OS within an LPAR.

8.5.1 Creating an HA infrastructure

WOLA supports both connections inbound to CICS and outbound from CICS. The following section summarizes the high-availability support for each configuration.

WOLA inbound to CICS

There are two high-availability components provided when connecting to CICS from WOLA:

- Connection factory failover

  This function is provided as part of the JCA support in WebSphere Application Server V8. The WebSphere Application Server high-availability support provides the ability to specify an alternate connection factory JNDI name in the connection factory pool custom properties. Each connection factory can optionally specify an alternate connection factory JNDI that will be used if the target CICS region is not available (Figure 8-12).

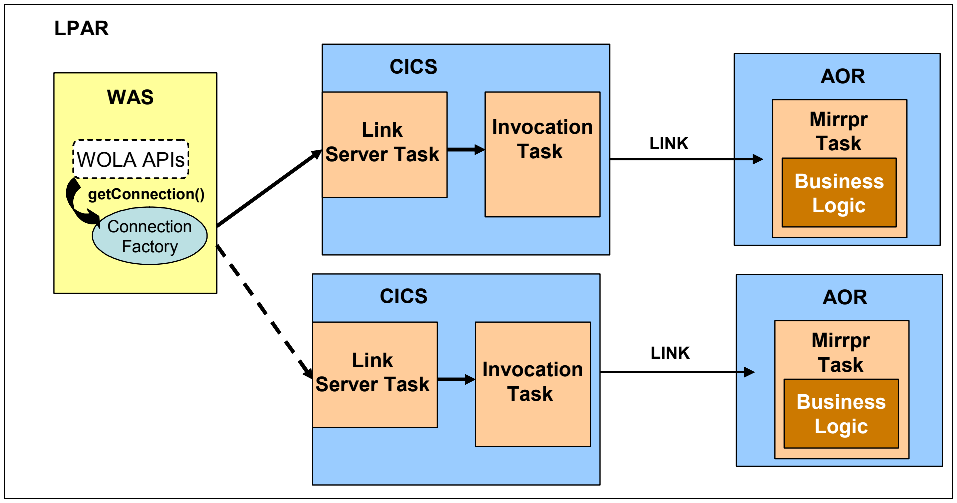


Figure 8-12 WOLA: Connection factory failover

The WOLA resource failover process is triggered when an application makes a getconnection() request for a resource (such as CICS) that has failed. When this situation occurs, a pre-defined alternate connection factory naming a backup CICS system is then used for the getconnection(). The number of failed attempts before the failover is tried can be set using the failure threshold property in the connection pool. In addition, WOLA uses a polling mechanism to detect when a primary resource is available again and will automatically start using the primary connection factory CICS system when it has successfully started and registered with the WOLA run time.

- Round-robin request distribution

  This provides the ability to distribute requests across multiple CICS regions. It must be enabled using a specific WOLA environment variable, and once enabled distributes requests in a round-robin way across all CICS regions that have registered using the same service registration. However, if requests are within the same transactional scope, they will always be directed to the CICS region first used, until the transaction is committed.

WOLA outbound from CICS to WAS

There is no high-availability functionality when making outbound calls from CCIS to WAS using WOLA, and so if the WAS system is not available, calls from CICS to WAS will not succeed.

Outbound calls do, however, support the usage of WLM for the prioritization of work in WAS. When using this function the WLM priority from the calling CICS transaction can be used to set the WLM service class for the EJB invoked in WAS.

8.5.2 Scaling

WOLA provides an efficient cross-memory mechanism for calls both inbound to WAS z/OS and outbound from WAS z/OS. It it designed for use with tightly coupled applications that make frequent or rapid calls between components.

8.5.3 High-availability considerations for WOLA

The following questions summarize the key high-availability considerations when using WOLA:

- Does WOLA support CICS dynamic routing?

  Yes, dynamic routing of CICS DPL requests can be used for inbound calls to CICS using WOLA. However, the invocation task and the link server task must run in the same CICS region that is registered with WOLA.

- Under what circumstances will WOLA failover to a backup?

  The WAS V8 connection factory failover function redirects requests to the backup CICS region as soon as a specified number of JCA calls fail with a ResourcException. In addition, new connections automatically use the primary CICS region as soon as it successfully re-registers with WAS, although existing connections continue to the backup CICS region.

- Can WOLA be used to route requests between LPARs in a sysplex?

  No, WOLA uses a cross-memory transport, and as such the WAS sub-system and the CICS region must be running in the same LPAR. If its necessary to route requests across a CICSplex environment, you can use CICS dynamic routing of DPL requests to achieve this. See “DPL routing” on page 230 for more details.

8.6 CICS web support

CICS provides built-in support for communications over HTTP. In this section we discuss the high-availability options for CICS web support and the key factors that affect scalability.

Note: CICS web support is also used as the HTTP transport for CICS ATOM feeds and for CICS Web service requests.

8.6.1 Creating an HA infrastructure

When creating a highly available infrastructure for CICS web support the issues should discussed in this section should be considered.

IP connection balancing

When accessing CICS applications using CICS web support, IP connection balancing can be used to provide a simple means of distributing requests across multiple listener regions running within the sysplex. Both port sharing and Sysplex Distributor can be used to distribute requests within or across LPARs in a sysplex. For more details refer to “Port sharing” on page 222 and “Sysplex Distributor” on page 222.

Note: CICS TS V4.2 provides a new MAXPERSIST option to control the number of persistent HTTP connections allowed for each HTTP listener. This is controlled via a predefined limit on the TCPIPSERVICE resource and is designed to be used to prevent any one listener region from becoming permanently overloaded beyond a certain threshold.

Optionally, WLM health metrics can also be used by port sharing or Sysplex Distributor to affect the distribution of new socket connections across cloned CICS listener regions. This is recommended in these cases:

- WLM service classes are used to prioritize the dispatching of transactions within the CICS regions using CICSPlex SM goal mode routing algorithms.
- Requests are dispatched under the same transaction ID in the listening regions and AORs.

SSL session-ID reuse

SSL session ID reuse allows an HTTP client and server to communicate with a shortened SSL handshake by allowing the client to reuse an SSL session ID without re-negotiating encryption keys with the server. When using CICS SSL support, session ID re-use is enabled using the SSLDELAY SIT parameter, which allows CICS to store and reuse a session ID for a certain period. Optionally, SSL session-IDs can also be stored in a sysplex wide cache. This can provided a significant performance advantage if SSL connections are used with IP connection balancing across multiple CICS regions (Figure 8-13).

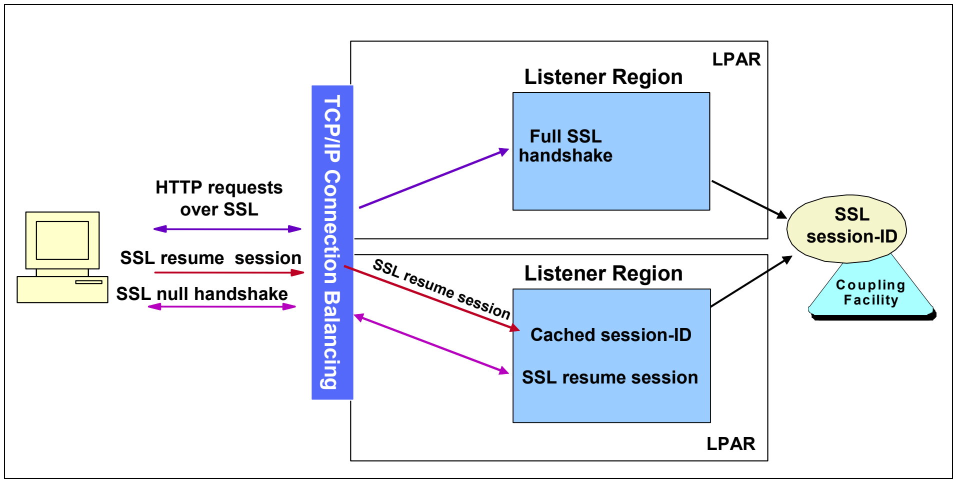


Figure 8-13 CICS web support: SSL session ID reuse

CICS dynamic routing

CICS does not provide the ability to route the web alias transaction (CWBA) from a listener region to an AOR. Thus, any program that issues CICS **WEB** commands must run within the listener region that handles the HTTP communication. Instead, if dynamic routing is required, we recommend that when the web application invokes the business logic in a CICS application, the LINK to the business logic is dynamically routed to an AOR using a DPL (Figure 8-7 on page 231).

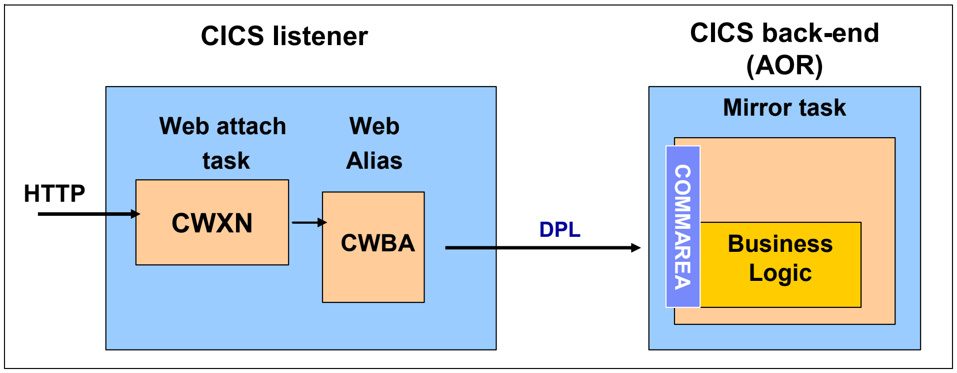


Figure 8-14 CICS web support: Dynamic routing

8.6.2 Scaling

CICS web support provides a highly efficient infrastructure for sending, receiving, and marshalling HTTP requests from directly within CICS and can support up to 65,000 concurrent connections due to the support for asynchronous TCP/IP receives. The key factors affecting the scalability of a CICS web support solution are usually the payload size of each request and the reuse of connections:

- Payload size

  The size of the payload is the principal factor affecting the performance of HTTP requests into CICS. The obvious rule is to keep your payload small and simple. However, in the real world, you do not always have the luxury of adhering to this rule. Larger messages result in higher CPU usage in both CICS and the IP stack and additional storage requirements. CPU usage typically increases linearly with payload size (Figure 8-15). The goals should be an awareness of these impacts with an aim to minimize the size of the input and output messages.

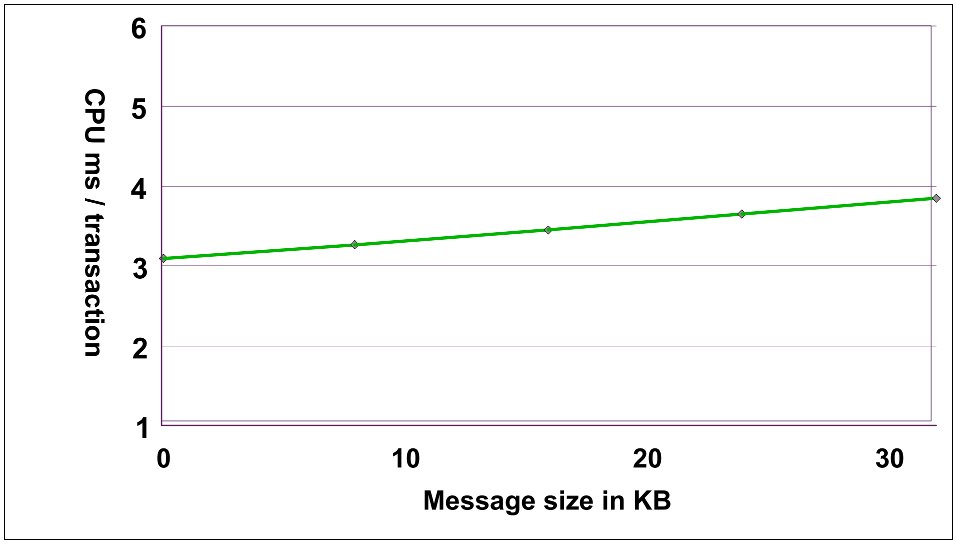


Figure 8-15 CICS web support: CPU usage versus message size

- Connection reuse

  When HTTP is used for inbound requests to CICS, persistent connections outperform non-persistent connections. The reason for this is that when using persistent connections, the client can reuse both the underlying socket connection and the CWXN task in CICS for subsequent messages. If the connection is not persistent, then the client will have to establish a new connection and also cause a new CWXN transaction to be created for every message. As such, we recommend setting SOCKETCLOSE on the TCPIPSERVICE definition to NO to take full advantage of the benefits of connection pooling.

When HTTP is used for outbound requests from CICS, then CICS TS V4.2 can now cache HTTP connections so that the dormant connection can be reused by any application that connects to the same host and port. This is controlled via the URIMAP resource and can significantly improve the scalability of web, web service, or EP applications using HTTP for outbound requests.

In addition to payload size and connection reuse, the method of mapping URI to resources and the usage of SSL are also factors affecting scalability:

- *URIMAPs* should be used to map HTTP requests to CICS resources and set the alias transaction. If an analyzer program is required for specialized security or adding functions, then avoid performing any processing that is likely to invoke delays, such as allocating storage, ENQs, or I/O requests.
- When using *SSL* the overheads of encryption can be substantially reduced by pooling connections, re-using SSL sessions, and exploiting encryption algorithms that are supported by the System z cryptographic hardware.

  In addition, when using SSL connections there is a pool of S8 TCBs used for SSL processing. These are used by CWXN or the alias task for the SSL handshake and to encrypt and decrypt data being sent and received. The S8 TCBs are used for the duration of the function that they are performing and returned to the pool afterwards. However, if a large number of concurrent SSL sessions need to be used, then the pool size can be increased up to a maximum of 1024 per region.

8.6.3 High-availability considerations for CICS web support

This section summarizes the key decision factors affecting a high-availability configuration with CICS web support.

How to route requests to AORs

DPL is the only available choice for routing work from web listening regions to AORs and can be used together with CICSPlex SM dynamic routing to distribute work across multiple AORs.

Tip: If DPL requests are issued from the web alias to an AOR and it is required to keep a consistent transaction ID across the listener and AOR, then it might be necessary to use the XPCREQ user exit to propagate the transaction ID into the AOR.

How to ensure CICS listener regions are not overloaded

When using IP connection balancing across multiple listener regions in conjunction with HTTP persistent connections, it is possible that individual CICS regions can become overloaded with too many HTTP connections, while others remain less well utilized. The CICS TS V4.2 TCPIPSERVICE option to control the number of persistent HTTP connections for each listening HTTP port can be used to address this situation by setting a maximum limit on the number of persistent connections per listener. This ensures that any connections beyond this limit are either redistributed to other listeners or used as non-persistent connections.

How to handle failback for restarted CICS listeners

If CICS listener regions are restarted when IP connection balancing is in use, then they might not be allocated new IP connections if persistent connections are in use. The best solution for this is to ensure that HTTP persistent connections are periodically recycled by setting a timed value for the SOCKETCLOSE setting on the TCPIPSERVICE so that connections are periodically recycled.

When to use WLM feedback with IP connection balancing

WLM feedback to Communications Server port sharing or Sysplex Distributor is recommend in these cases:

- WLM service classes are used to prioritize the dispatching of work within the CICS regions according to response time or velocity goals.
- Tasks run under the same transaction IDs in the listening region and AORs.

8.7 WebSphere MQ

This section discusses how to create a highly available messaging infrastructure when using WebSphere MQ, and the implications of this on the scalability of the system. For further details about creating WebSphere MQ high-availability configurations refer to *WebSphere MQ in a z/OS Parallel Sysplex Environment*, SG24-6864.

8.7.1 Creating an HA infrastructure

When WebSphere MQ is used as the transport there are a variety of high-availability options that can be used to distribute messages across different queues. The two principal mechanisms are *clustering* and *queue sharing groups,* which can also be used together with IP connection balancing solutions.

Clustering

A cluster is a network of queue managers that are logically associated (Figure 8-16). Clustering both simplifies system administration and allows you to define instances of the same queue on more than one queue manager. This later ability provides a *push distribution* high-availability solution. Messages for a clustered queue are distributed throughout the actual queue managers in the cluster using a simple routing algorithm, such as round robin. The target queue managers can be running at different locations and on different platforms.

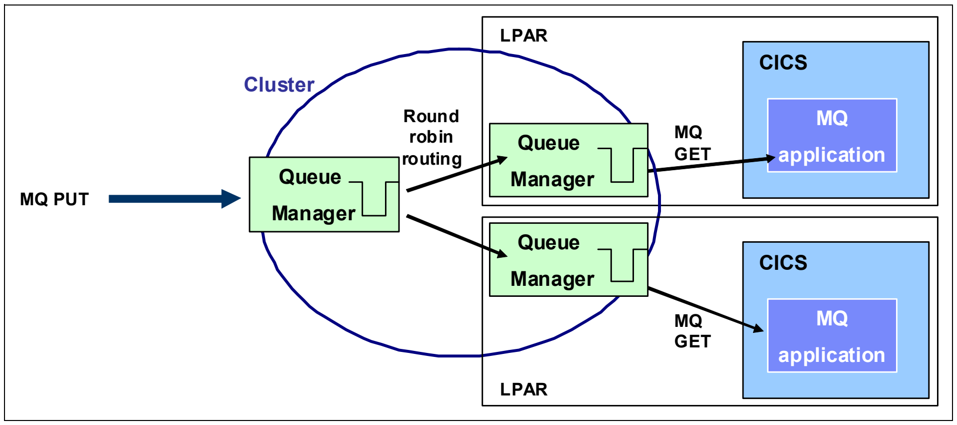


Figure 8-16 WebSphere MQ clustering

Queue sharing groups

Queue sharing groups are a unique feature of WebSphere MQ for z/OS that use the functionality of the Parallel Sysplex coupling facility and DB2 data sharing groups to provide a sysplex-wide shared queue model. Any queue manager in the group can service any shared queue and can continue processing a queue if a queue manager in the queue-sharing group fails. WebSphere MQ detects whether a queue manager disconnects from the coupling facility abnormally and, where possible, other queue managers in the group perform peer recovery to complete pending units of work for that queue manager. Shared queues are supported by CICS TS V4 and can be used by both the CICS-supplied trigger monitor or the CICS DPL bridge. In addition, shared queues can be exploited for both inbound messages and outbound messages.

There are two typical CICS high-availability models used for queue sharing, trigger every and trigger first or depth, based on the mechanism for writing trigger messages to the initiation queue. In the trigger first/depth model a trigger message is written to the initiation queue after the message level in the request queue has reached the specified depth, whereas in the trigger every model a message is written to the initiation queue for every message arriving on the queue. Typically, trigger first is used to start long-running transactions, and trigger every when the message rate is low, such as a few messages a second.

In the trigger first or depth scenario (Figure 8-17) the initiation queue is usually defined as private in each queue manager. This ensures that when messages arrive on the request queue, trigger messages will be created in each initiation queue in each queue manager. This causes the CICS trigger monitor transaction (CKTI) in each CICS region to start a transaction to process messages until the queue is empty. This model is typically used for long-running applications that need to be triggered at the start of business processing, and is more efficient than trigger every, as less trigger messages are created and fewer CICS transactions started. However, if multiple CICS regions in the same LPAR are configured to use the same initiation queue, then it does not provide a way of distributing work across these regions, as only one CKTI transaction will be invoked to process the request queue.

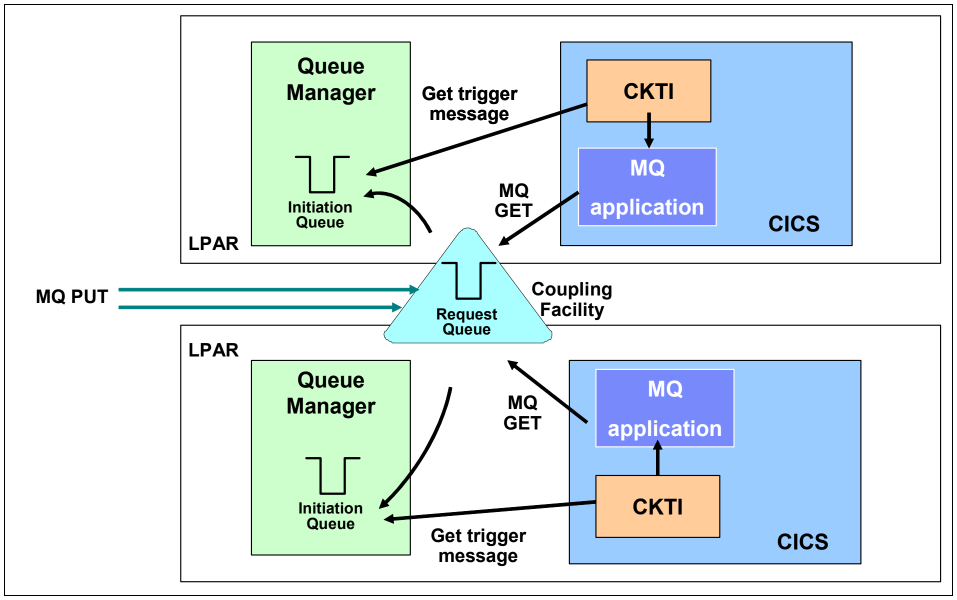


Figure 8-17 Queue sharing groups: Trigger first or depth

With the trigger every model, a trigger message is written to the initiation queue for every message that arrives on the request queue (Figure 8-18). If the initiation queue is shared across the sysplex, then one CKTI transaction will be triggered for each message that arrives on the request queue. This model ensures that as many MQ applications that are started in CICS as messages arrive on the shared request queue, and this helps to improve the distribution of work across large numbers of CICS regions. However, it is not as efficient as trigger first/depth because some CICS transactions are likely to be started that do not consume messages, and when used with high message rates can result in the flooding of CICS with too many active transactions.

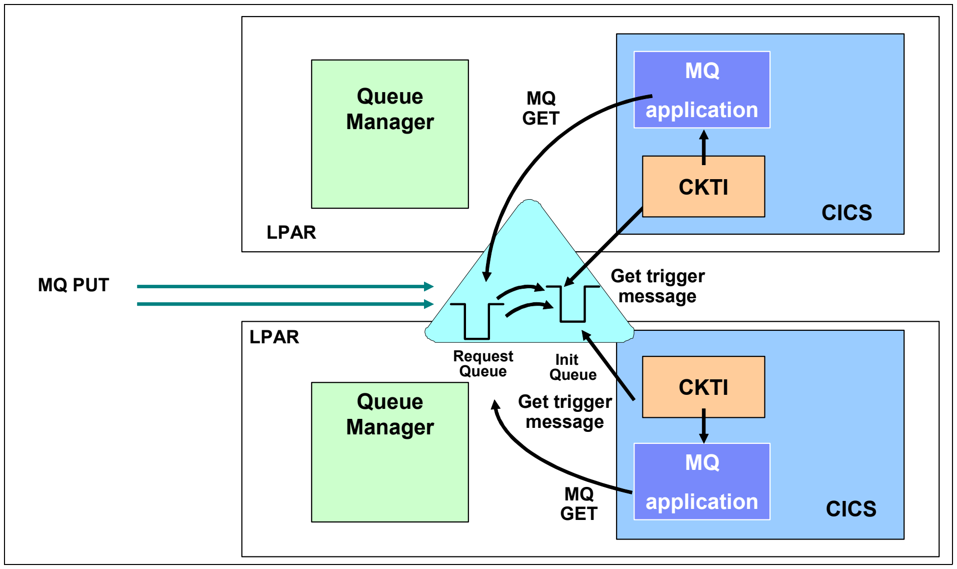


Figure 8-18 Queue sharing groups: Trigger every

With the *trigger every* model a trigger message is written to the initiation queue for every message that arrives (Figure 8-18). In this scenario the initiation queue is usually shared so that only one CICS trigger monitor transaction (CKTI) will be triggered for each message that arrives. This model ensures that as many WebSphere MQ applications are started as messages arrive on the shared request queue, and thus allows workload to be distributed across a large number of CICS regions. However, it is not as efficient as alternative models because some applications are likely to be started that do not consume messages, and thus it is recommended for low-volume messaging.

IP connection balancing and shared channels

Shared channels are a feature of the WebSphere MQ channel initiator that enables you to use IP connection balancing technology to distribute incoming connections across a set of eligible queue managers. If any queue manager within the queue sharing group fails, remote queue managers are still able to connect to the queue sharing group through a shared channel and put their message on to the shared queues.

Queue sharing groups can be used in tandem with Sysplex Distributor (Figure 8-19). In this scenario, a queue manager can be cloned across a set of LPARs and listen on a single shared virtual IP address to receive connections from remote queue managers.

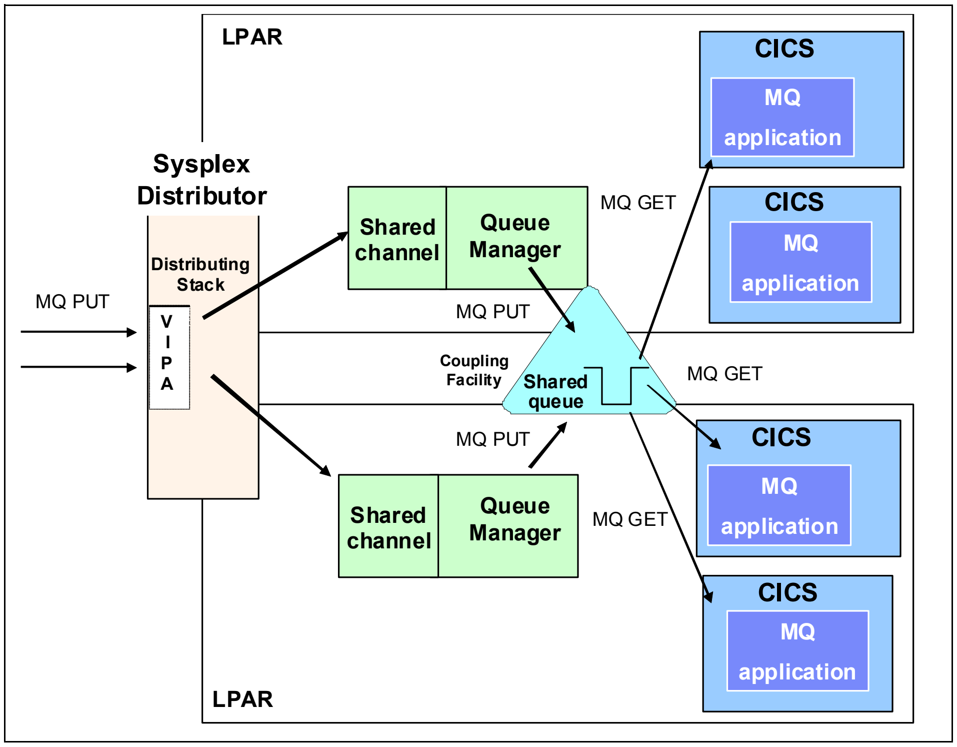


Figure 8-19 Queue sharing groups with IP connection balancing

8.7.2 Scaling

WebSphere MQ provides a highly scalable infrastructure for asynchronous messaging, which can be exploited by applications running within CICS. These are the key considerations in terms of scaling:

- Payload size

  The larger the payload the higher the overhead in terms of both CPU and network I/O. If using shared queues, messages over 63 KB in size require the use of DB2 as a message store, which will have a higher overhead than using the coupling facility that is used for smaller messages.

- Threadsafe

  MQ GETs can be issued from a threadsafe CICS program and can run on an L8 CICS TCB. However, `CICS START` commands are not threadsafe, so any STARTs issued will cause a TCB switch to the QR TCB.

- Persistent messages

  These have a higher overhead compared with non-persistent messages due to the logging and recovery requirements associated with message persistence.

- How many queue managers are required

  Typically, one queue manager per LPAR is usually sufficient, and this can be shared by multiple CICS regions. However, multiple queue managers can be used, and they can listen on a shared port to provide both redundancy and scalability.

8.7.3 High-availability considerations for WebSphere MQ

This section summarizes the key decision factors affecting a high-availability configuration with the CICS and WebSphere MQ.

Clustering versus queue sharing

Clustering provides a push distribution model, is simple to implement with lower operational overheads, and is supported across all WebSphere MQ platforms. However, it is does not guarantee full availability, as in-flight messages can get stuck either in the transmit queue or the local queue on the target queue manager if the queue manager fails.

Queue sharing provides a pull distribution model, which provides the advantage that the performance of the consuming system can dictate how many messages will be processed. It also has the advantage that it provides peer recovery for in-flight messages, which removes the window of failure during queue manager failure, and the same infrastructure can be used for inbound and outbound message distribution. However, queue sharing is more involved to configure and requires the usage of a coupling facility and a DB2 data-sharing group.

How to handle high-volume queues

When using a queue with a high message arrival rate, the messages might arrive faster than a single CICS transaction can consume them. In this case it is often best to not use triggering, but instead to start an instance of the processing application in each CICS region at startup using an infinite get-wait. If the volume is consistently heavy and not a daily cyclical issue, this option is effective and less expensive overall than triggering. However, there are also several potential triggering-based solutions that can be considered:

- Develop a trigger monitor that analyzes queue depth or the number of open handles and starts additional instances of the processing application as required, either on the same region or on a remote region.
- Create a processing application that starts a new CICS transaction for every MQ GET issued from the request queue. Any further processing of the messages can then be performed in the started transaction rather than congesting the original message-consuming application. This is a good option if there is extensive message-processing performed in the message-consuming application.
- Create a processing application that, after reading a fixed number of messages, restarts a new instance of itself and ends. This allows CICSPlex SM to route the work to another CICS region.
- If your CICS applications are threadsafe, create a trigger monitor capable of starting multiple processing application instances at once. This can improve parallelism and thus help speed draining of the request queue.
- For a working example of a pattern for implementing some of these techniques, refer to the following IBM developerWorks® article:

  http://www.ibm.com/developerworks/websphere/library/techarticles/0511_suarez/0511_suarez.html

8.8 CICS sockets

CICS sockets provides a low-level interface for interfacing with CICS applications using the TCP/IP socket protocol. For further details refer to *z/OS V1R12 Communications Server IP CICS Sockets Guide,* SC31-8807.

8.8.1 Creating an HA infrastructure

There are two listener models provided to create CICS sockets applications:

- Iterative server

  This is the simplest model and provides in-line processing of the socket and the calls to the CICS business logic. Because there is only one transaction serving the socket, all the messages sent over the socket are processed serially in the same CICS task (Figure 8-20).

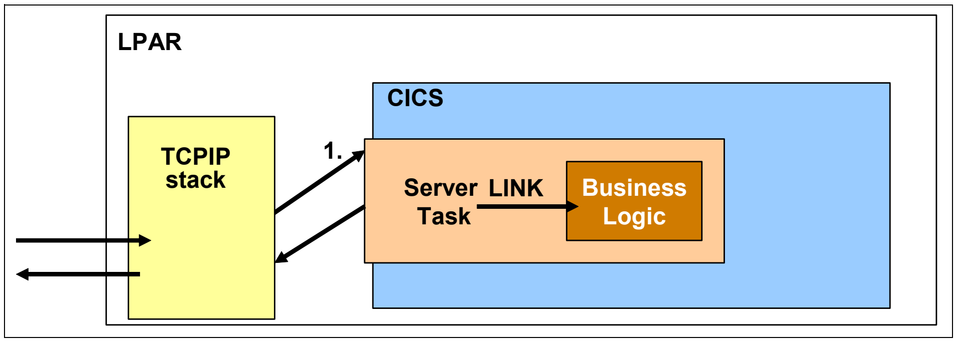


Figure 8-20 CICS sockets: Iterative server

- Concurrent server

  CSKL is the supplied concurrent server transaction and starts child server transactions for every message received. Different child server transactions can be invoked depending on the pre-defined formats of the incoming messages, and multiple instances of child server tasks can run in parallel to process multiple sockets (Figure 8-21).

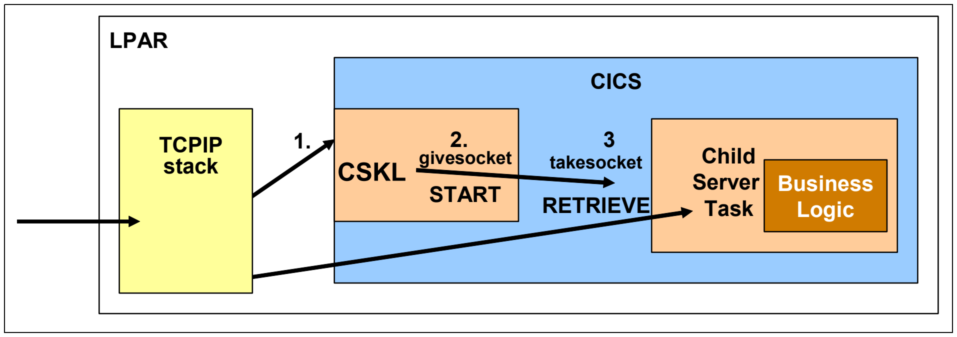


Figure 8-21 CICS sockets: Concurrent server

IP connection balancing with CICS sockets

CICS sockets supports IP connection balancing and can be used with port sharing and Sysplex Distributor to balance incoming sockets across multiple CICS regions.

CICS dynamic routing

CICS sockets support two different models for dynamic routing of work to CICS AORs as follows:

- Routing STARTs and givesocket

  This model is supported with the concurrent server and works as shown in Figure 8-22. The child server task is started in the AOR by defining the transaction as remote. This function ships the `START` command to the AOR, which then takes ownership of the original socket using the `takesocket` command. The child server task can be routed to any region in the same LPAR that is bound to the same IP stack as the listener region.

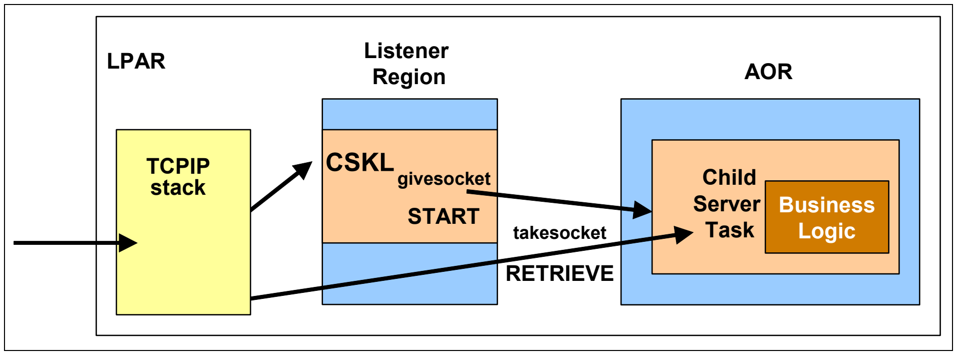


Figure 8-22 CICS Sockets: Routing STARTs

- DPL

  Dynamic Program Link is supported with both the concurrent server and the iterative server model and works as shown in Figure 8-23. In this scenario the routing occurs using DPL when the server task links to the target business logic program passing a COMMAREA or CHANNEL. For further discussion of the routing models refer to “What CICS routing model should be used” on page 258.

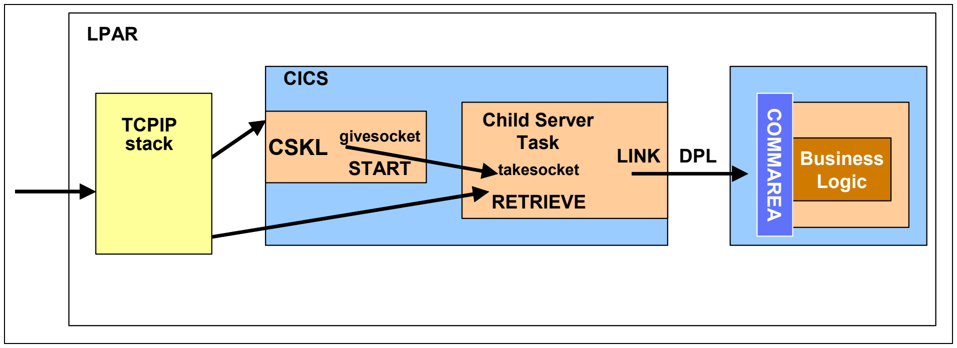


Figure 8-23 CICS Sockets: DPL

8.8.2 Scaling

CICS sockets is a low-level programming interface, and as such the qualities of service are very much dependant on the application design. These are key factors affecting the scalability of the solution:

- Usage of the concurrent or iterative listener
- Ensuring that the CICS sockets listener transaction has the highest dispatching priority within the CICS region

8.8.3 High-availability considerations for CICS sockets

This section summarizes the key design questions when creating an HA infrastructure for use with CICS sockets

What CICS routing model should be used

The DPL routing model is the recommended routing model and provides the benefit that all the socket handling code is run in the listener region and the DPL can be routed to any CICS region in the CICSplex. The START routing model can be beneficial if the listening region becomes overloaded. However, it is limited to the same LPAR and requires additional setup of the CICS sockets TRUE in the AOR.

How to route long-running tasks

CICS does not provide a built-in mechanism to allow STARTed transactions to return data back to the parent task that initiated them. This means that if multiple requests are being sent over a persistent socket connection, then a more complex application design is required to allow the STARTed child task to return data back to the listener transaction. CICS offers a range of facilities to assist with this scenario, including usage of Transient Data Queues (TDQs) or Business Transaction Services (BTS).

How to handle multiple socket application

Either define a concurrent listener on the same port to start a different transaction based on the format of the incoming message or define a different listener task (with unique transaction IDs) that invokes different child server transactions.

Part 3

Integration scenarios

In Part 3 we summarize three customer-based CICS integration scenarios. For each of the scenarios we provide the customer context, describe the specific project requirements, and highlight the strengths of the chosen solutions. The scenarios cover the following CICS integration technologies:

- CICS Web services

  In this scenario, we show how web services can be used to create a loosely coupled solution in which CICS services can be invoked from a variety of different platforms.

- CICS Transaction Gateway for z/OS

  We describe how the CICS TG for z/OS provides a transactional connector from the Java EE environment. We also focus on the techniques that can be used to create a high-availability configuration.

- Messaging

  In the final scenario, we look at how a messaging infrastructure based on shared queues can be used to create a highly responsive and robust solution for credit card processing.

9

CICS Web services scenario

This chapter looks at the example of a bank that wants to extend some of its CICS core banking services such as account transfers and posting inquiries to internal distributed systems and selected business partners. We review how the implementation of CICS Web services and the DataPower XI50z improves the bank's business agility, reduces IT costs, minimizes risk, and simplifies IT complexity.

9.1 Objectives

Many banks have embarked on a core banking transformation strategy to gain flexibility and reduce cost. A service-oriented architecture (SOA) is normally the preferred architecture because it facilitates maximum reuse of existing assets.

Web services can be used to enable CICS integration based on a common set of standards covering message format, protocol, and security. An ESB pattern provides the greatest flexibility, providing the ideal location for functions such as data transformation, protocol switching, and enforcement of security policy. The DataPower XI50 appliance provides these ESB capabilities with an ease of use and performance that is unmatched by other ESB solutions.

Solution requirements

It is the main functional requirements of the project that led to the choice of using CICS Web services support with the DataPower XI50. Table 9-1 shows how CICS Web services support meets the major functional requirements of the project.

Table 9-1 Main functional requirements

Functional requirement	Solution
Ability to invoke CICS Web services from any platform (including J2EE and .Net)	Web services are platform neutral.
Ability to support a variety of different authentication mechanisms	DataPower provides an extensive set of authentication options out of the box.
Ability to implement an end-to-end security solution based on identity propagation	CICS Web services and DataPower support z/OS Identity Propagation.
Ability to distinguish requests between internal requests and external requests	DataPower is able to apply different security policies dependent on the type of service requester.
Bi-directional support (CICS can be service provider or service requester)	CICS Web services support is bi-directional.
Support for long messages	CICS Web services and DataPower support the use of Message Transformation Optimization Mechanism (MTOM) for long messages.
Ability to publish CICS services to a service registry	CICS Web services can be published to the WebSphere Services Registry and Repository (WSRR)

These are the main non-functional requirements of the project:

- Minimized risks

  The business-critical web services must be available at all times to authorized users. The infrastructure must be able to support continuous service availability across planned and unplanned outages.

- Optimized performance and scalability

  The solution must be optimized and must meet the performance and scalability expectations of the customer. One of the prime objectives is to demonstrate how a CICS Web services workload can be dynamically managed based on performance goals.

- Simplified configuration and infrastructure management

  Services need to be enabled quickly and the SOA infrastructure must be easy to manage.

- Real time monitoring

  Detailed real-time information (including service hit rates, response times, and message lengths) must be available for problem diagnosis. See 9.3.4, “Real-time monitoring” on page 271 for information about how we monitored the CICS Web services.

In 9.3, “Implementation” on page 266, we take a brief look at how the non-functional requirements of the project were addressed.

9.2 Architecture

CICS Web services can be accessed directly from a service requester or indirectly via a service bus. A service bus approach has several advantages:

- Service requesters do not need to have knowledge of the specific physical location of service providers. The service is *virtualized*.
- The service bus can manage the security characteristics of inbound or outbound web services requests, changing transports, and performing security tasks such as authentication and identity mapping, and propagating identities to the target service provider
- The service bus can simplify the management of service deployment and configuration.

While there are a number of service bus implementations that meet the above requirements, the choice was made in this project to use the DataPower XI50 as a service bus component (Figure 9-1).

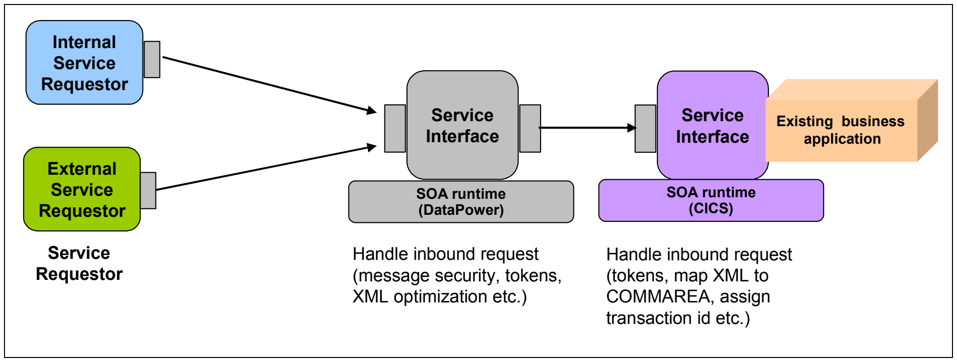

Internal Service Requestor
External Service Requestor
Service Requestor
Service Interface
SOA runtime (DataPower)
Handle inbound request (message security, tokens, XML optimization etc.)
Service Interface
Existing business application
SOA runtime (CICS)
Handle inbound request (tokens, map XML to COMMAREA, assign transaction id etc.)

Figure 9-1 CICS Web service virtualization

9.3 Implementation

The DataPower XI50 is a purpose-built SOA appliance for delivering highly manageable, security-enhanced, and scalable SOA solutions. As specialized SOA hardware, it provides many core functions to SOA deployments in a hardened device including integrated Enterprise Service Bus (ESB) capabilities, data enablement and integration features, and the capacity to improve web services management and SOA governance.

The DataPower XI50z provides the same application functionality as the XI50. What makes the DataPower XI50z different is the physical integration within the zEnterprise system. The DataPower blade is installed as an optimizer within the zBX. For zEnterprise customers, this provides additional value opportunities in areas such as security and extended System z integration. In addition to the physical integration within zEnterprise, the DataPower XI50z benefits from the integrated management provided by Unified Resource Manager. The DataPower XI50z blades are managed as part of the zEnterprise *ensemble*.

Figure 9-2 shows the bank's chosen deployment pattern for the DataPower XI50z.

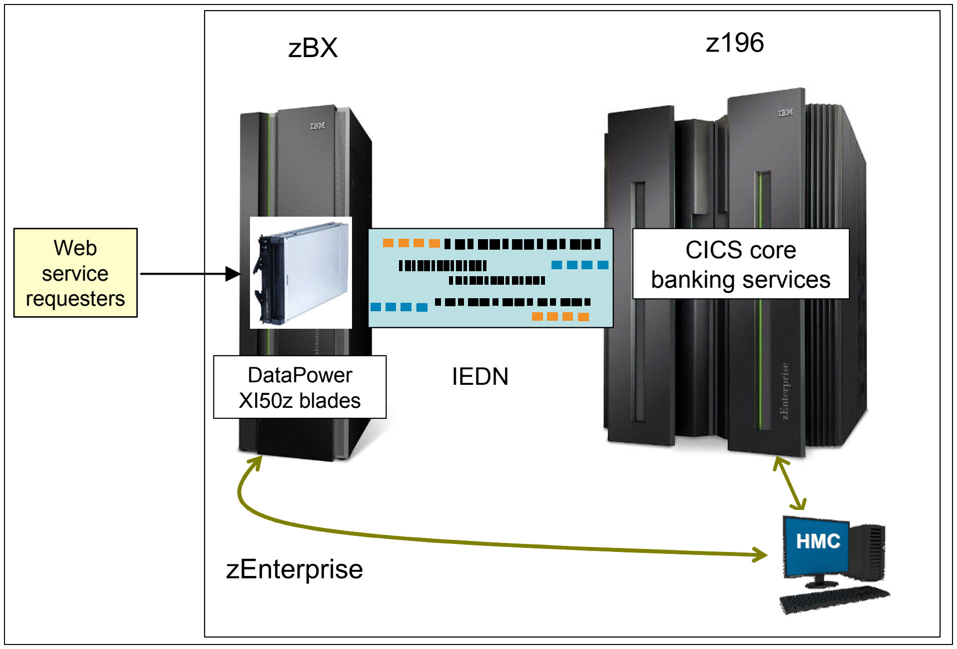


Figure 9-2 DataPower XI50z deployment

The DataPower XI50z is an ideal choice for an SOA Gateway or ESB when the target services are traditional z/OS assets such as CICS and IMS applications, data residing in DB2 for z/OS, or other services deployed on virtual servers within the zBX. These service interactions then benefit from the fast secure Intraensemble Data Network (IEDN) connecting the virtual servers within the ensemble.

9.3.1 Minimizing risks

Deploying the DataPower XI50z as an SOA Gateway minimizes the bank's risks in the following ways:

- It enables tight integration between the bank's security domains and user registries, thus minimizing the risk of unauthorized access to the core banking systems.
- It minimizes the risk of outage by benefiting from the unparalleled availability and workload management capabilities of zEnterprise.

End-to-end security

The bank has to prepare for stringent compliance regulations, which dictate that all service invocations must be audited and that the originating user's identity must be included in the audited record. The current security model does not fulfill this requirement because when the distributed identity is mapped to a RACF user ID, the originating user's identity is lost.

The z/OS identity propagation support available with CICS TS V4.1 and z/OS V1.11 solves the bank's security challenge. The DataPower XI50z authenticates the credentials supplied by the client and maps them to a z/OS-specific ICRX token that contains the distributed identity of the user. The request is then forwarded to CICS over the secure IEDN and CICS passes the token to RACF so that the client's identity can be mapped to a RACF user ID. The advantage of this solution is that the original caller's identity is not lost. It is stored as an extension to the RACF identity. See "z/OS identity propagation support with CICS Web services" on page 151, for more information about how CICS Web services support identity propagation.

High availability

When deploying CICS Web services in a parallel sysplex, you can take advantage of the z/OS-specific workload management capabilities, including Sysplex Distributor and MVS Workload Manager (WLM). Figure 9-3 shows the parallel sysplex configuration used for this project.

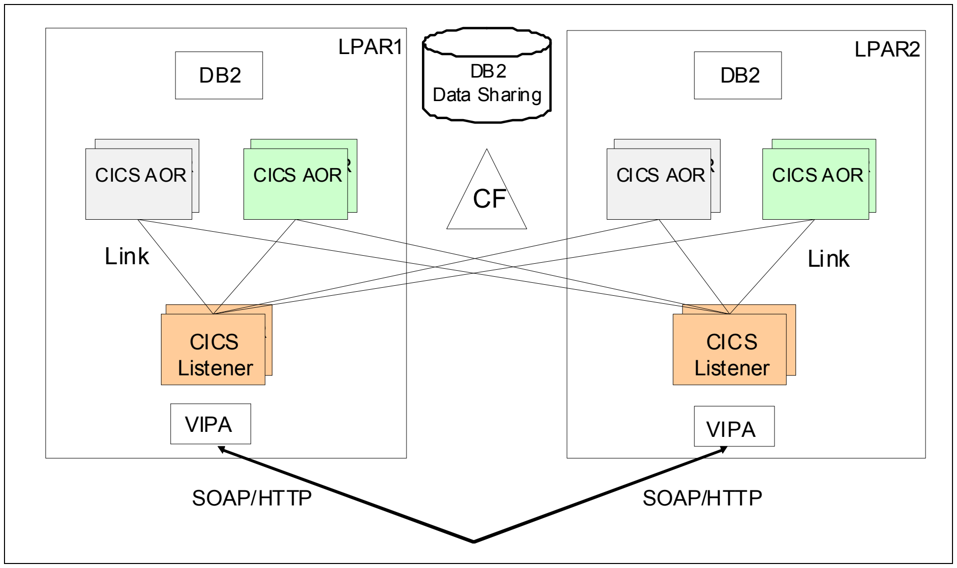


Figure 9-3 CICS Web services high availability configuration

Here we describe Figure 9-3 on page 268:

- Sysplex Distributor is used for workload management of TCP/IP connections across two LPARs (LPAR1 and LPAR2).
- Multiple CICS regions on each LPAR listen on a shared TCP/IP port.
- Program link requests are dynamically routed to cloned AORs using CICSPLex SM.
- CICS business logic programs running in the AORs share access to business data using DB2 data sharing.

The bank also chooses to use Sysplex Distributor to distribute requests across the DataPower blades within the zBX (Figure 9-4). This simplifies workload management and improves availability. It also allows the bank to remove the external workload distribution appliances that have been used previous to this solution.

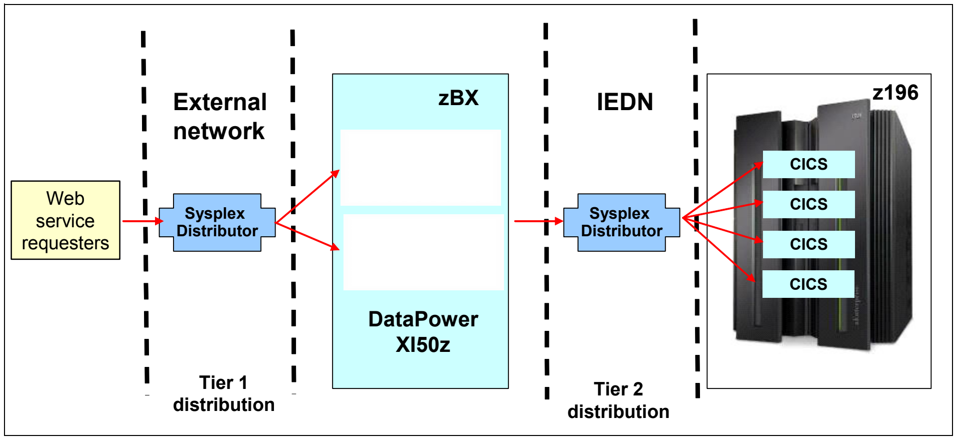


Figure 9-4 Using Sysplex Distributor for high availability

9.3.2 Optimized performance and scalability

The scalability of CICS Web service applications is primarily dependant on the size and complexity of the SOAP messages. The chosen security model can also have a big impact. For very long or complex messages, or for certain types of security solution, it can be optimal to front-end the CICS Web services with an SOA appliance.

Using an SOA appliance fits well with the bank's strategy to deploy different components of workloads on the *best fit* platform. DataPower is a clear best fit for heavy XML processing and security functions. For example:

- When DataPower is used to parse the SOAP body of very large messages, the bank's tests show that this reduces the processing cost in CICS by up to 75%.
- Using DataPower to validate XML signatures is shown to be more than five times more efficient than a software-based solution.

Using DataPower as an optimizer in this way frees up resources for other processing, such as transactional, data access, and business logic.

In addition, performing such processing in a special-purpose optimized blade has additional benefits for the bank in lowering power consumption, reducing overall cost, and improving scalability.

9.3.3 Simplifying configuration and infrastructure management

An appliance is easy to configure and allows solutions to be “dropped in” quickly, thus saving time and labor costs. Figure 9-5 shows the configuration differences between an appliance and a typical software-based ESB.

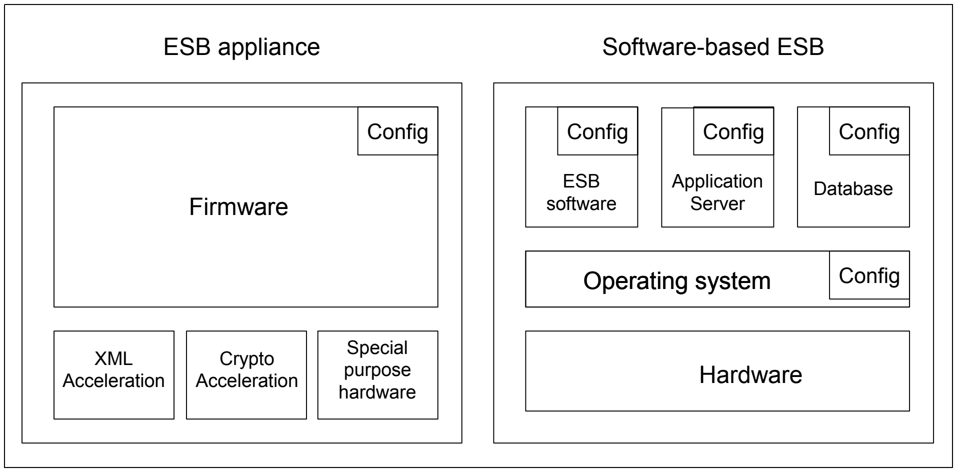


Figure 9-5 How an appliance simplifies configuration

An ESB appliance can save labor costs by providing one configuration point and no programming. A software-based ESB, however, typically has broader reach through a full programming model, but a wider set of configuration points.

The choice of the DataPower XI50z also fits with the bank's aim to minimize infrastructure management by managing different platforms in a consistent way. The zEnterprise Unified Resource Manager simplifies the bank's platform management by providing a single operations console. The Unified Resource Manager also:

- Monitors the health and energy consumption of the DataPower XI50z blade.
- Consolidates error logging across the ensemble, which consists of the DataPower XI50z blade and all resources (z196 and zBX components) that are part of the workload.
- Simplifies problem determination by providing *call home* support for current or expected problems.

9.3.4 Real-time monitoring

Another challenge faced by the bank is to efficiently monitor the services that it provides to its business partners and internal systems, in particular, to monitor against a set of pre-defined response time goals, to be able to identify a problem when it occurs and quickly identify the location and root cause of the problem.

The bank has extended its existing IBM Tivoli Monitoring infrastructure, which provides the enterprise infrastructure dashboard through the IBM Tivoli Enterprise Portal (Figure 9-6).

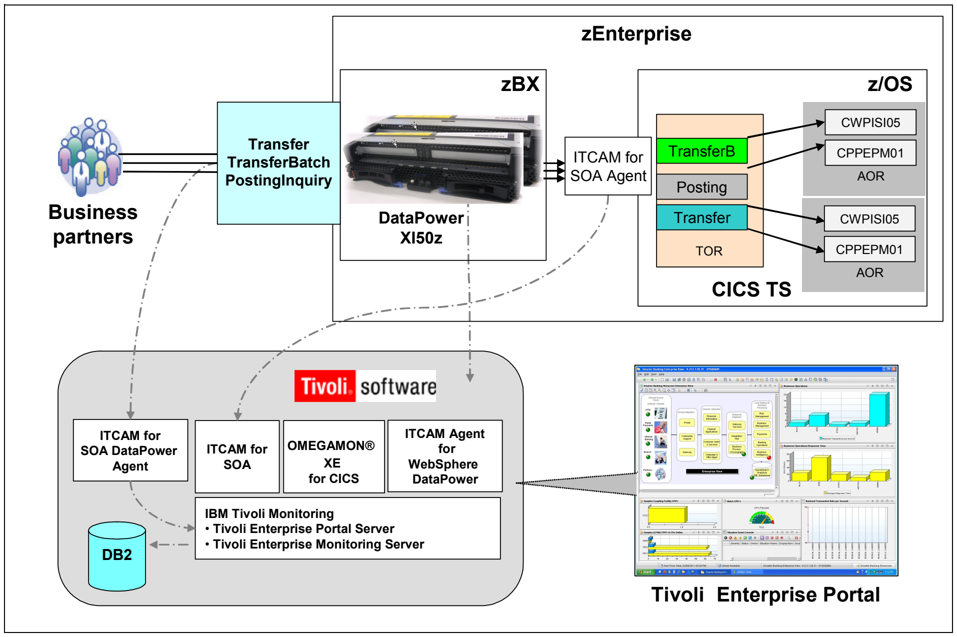


Figure 9-6 IBM Tivoli Monitoring infrastructure

Here we describe Figure 9-6:

- IBM Tivoli Composite Application Manager Agent for WebSphere DataPower Appliance is used to perform detailed monitoring of DataPower.
- IBM Tivoli OMEGAMON® XE for CICS is used for detailed analysis of web services in CICS, including tracking against service response-time goals.
- IBM Tivoli Composite Application Manager for SOA is used to monitor the end-to-end performance of the web services across both the CICS and DataPower runtime environments.

To set response time goals for web services, specific requests are associated with transaction identifiers using URIMAP resource definitions. These transactions are then classified by MVS WLM and assigned to a service class with a defined performance goal. This allows the response times for the CICS Web services to be monitored (Figure 9-7).

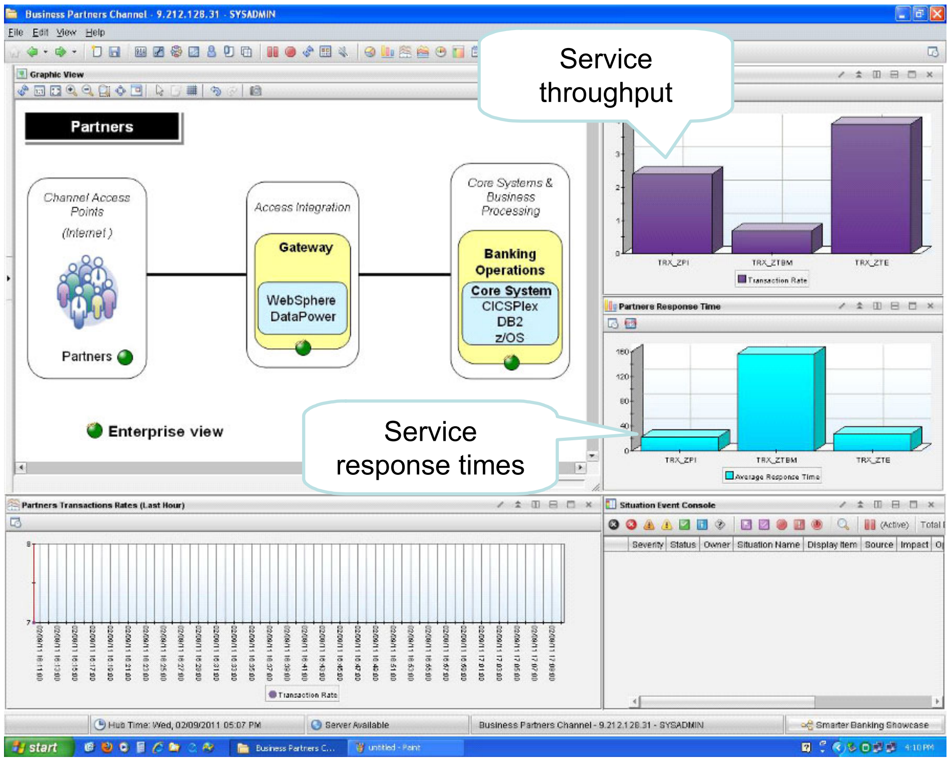


Figure 9-7   CICS Web service monitoring

9.4 Solution summary

CICS Web services provides the flexible integration and loose-coupling required by the bank. DataPower XI50z complements the use of CICS Web services by providing ESB capabilities that are easily configurable and the ability to offload specific CPU-intensive operations.

10

CICS TG for z/OS scenario

This chapter looks at a CICS integration architecture based on the CICS Transaction Gateway (CICS TG) for z/OS. We review how the implementation of CICS TG for z/OS addresses the requirements of TCP/IP-based access to CICS with high availability, while continuing to provide transactional integrity with CICS and DB2.

10.1 Objectives

Many customers have migrated their SNA networking infrastructures to TCP/IP, with the result that applications that depend on APPC programming interfaces have become obsolete. As such, modern IP-based alternatives are required that provide equivalent capabilities for SNA qualities of service, such as two-phase commit, high availability, and security. The architecture described in this chapter addresses the requirements in Table 10-1.

Table 10-1 Project requirements

Requirement	**Solution**
Migration from SNA to TCP/IP	► CICS TG support for TCP/IP client connectivity
No single point of failure	► TCP/IP Port Sharing and Sysplex Distributor with cloned Gateway daemons ► CICS TG dynamic server selection ► CICSPlex SM dynamic routing
High availability and scalability	► Cloning of Gateway daemons ► Dynamic server selection ► CICSPlex SM workload management
Ability to invoke a CICS transaction and update DB2 within the scope of a distributed unit of work	► XA support provided by the WebSphere Application Server EJB container and CICS TG
Integration with existing CICSPlex TOR to AOR routing configuration	► UOW affinity support for CICSPlex SM dynamic routing of DPL requests
Minimizing change to CICS applications	► Removing access to COMMAREA or channel-based CICS programs using the CICS TG External Call Interface (ECI)
Optimized performance	► CICS TG providing a highly scalable integration technology using binary formatted messages with optimized data transmissions and syncpointing

10.2 Architecture

The CICS Transaction Gateway can be installed on a wide variety of platforms and can be operated as a Gateway daemon in its own address space (remote mode), or it can be co-located within the application server (local mode). The solution described uses the CICS TG for z/OS running as a standalone Gateway daemon. This configuration has several advantages:

- Two-phase commit XA support is provided for any release of CICS TS for z/OS, allowing transactional updates to be coordinated across multiple resource managers, including multiple CICS regions and other enterprise information systems (EIS) such as DB2.
- High-availability configurations are simple to create and support is integrated with the CICS TG XA support.
- Secure SSL connections can be made directly into the Gateway daemon running on z/OS using the SSL support provided by the JSSE component of the IBM SDK for z/OS.
- Asserted identity configurations are supported, simplifying end-to-end security architectures.

Figure 10-1 shows the high-level architecture of the solution.

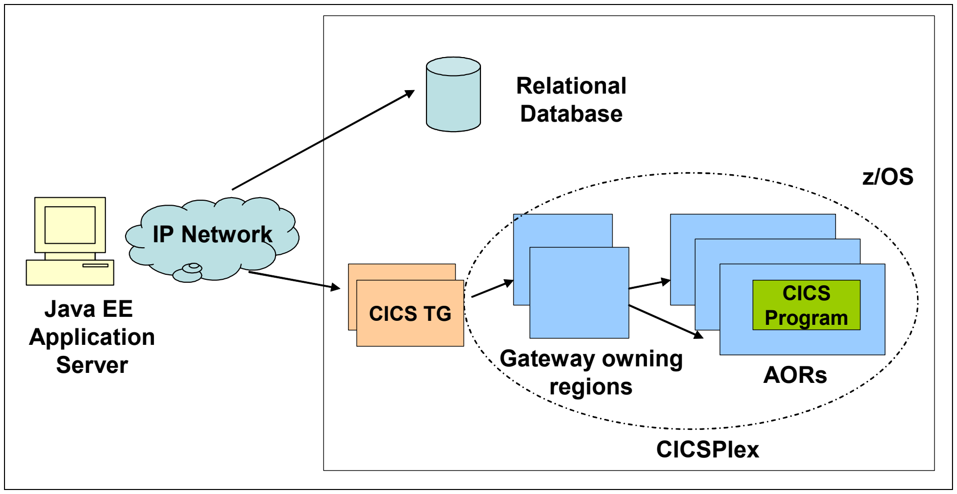


Figure 10-1 CICS TG scenario architecture

10.3 Implementation

The configuration consists a set of cloned Gateway daemons spread across multiple LPARs in a Parallel Sysplex, providing both scalability and availability. The Gateway daemons are able to process two-phase commit XA requests, enabling JDBC updates to be coordinated in the same distributed unit of work as calls to CICS programs (Figure 10-2).

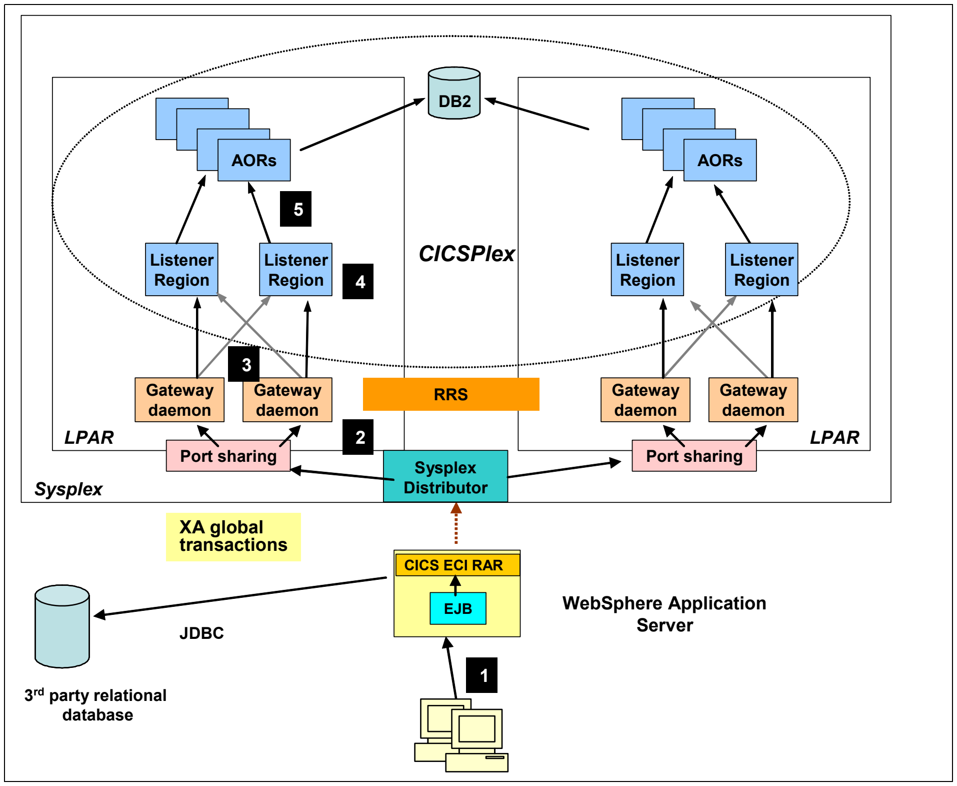


Figure 10-2 CICS TG z/OS high-availability configuration

The configuration used in Figure 10-2 consists of the following components:

- EJB components in WebSphere Application Server use the XA version of the CICS ECI resource adapter to make calls to CICS applications and JDBC calls to a third-party relational database in the same global transaction.
- Cloned Gateway daemons are part of a highly available Gateway group and listen on a shared port in each LPAR. Each LPAR has a dynamic VIPA used by the cloned Gateway daemons, to which incoming TCP/IP connections are distributed by Sysplex Distributor, providing TCP/IP connection balancing

across the LPARs. The number of cloned Gateway daemons can be increased to meet increasing system load, providing horizontal scalability.

- CICS TG dynamic server selection is used to control routing of ECI requests to CICS listener regions, and the CICS TG ensures that even in the event of failure all scenario requests in the same global transaction are routed to the same listener CICS region.
- Two CICS listener regions are configured per LPAR to provide high availability and to enable routing across a set of available AORs, allowing integration with existing CICSPlex SM workload management.
- CICSPlex SM UOW affinity support is used to ensure that all requests in the same distributed unit of work are routed by the listener CICS regions to the same AOR. This prevents any deadlock situations when the CICS AORs access the shared DB2 relational database.

10.4 Solution summary

The CICS TG for z/OS provides an easy-to-use and highly scalable TCP/IP-based connector technology for accessing CICS applications. This combined with the ability to coordinate transactions in CICS with updates to other XA capable resource managers, such as DB2, provides a simple way of re-using existing CICS assets and integrating them in a broader SOA framework.

11

Messaging scenario

This chapter looks at a CICS integration scenario based on a messaging infrastructure using WebSphere MQ. We review how the implementation of shared message queues creates a highly responsive and robust solution for credit card processing.

11.1 Objectives

Many banks have CICS-based credit card processing services, such as credit authorizations, that require real-time, near-instant decisions. This often poses challenges of scalability and throughput, especially during peak holiday season.

WebSphere MQ can be used to provide access to such programs using a request-reply pattern. The use of shared message queues enables this type of credit card processing solution to be highly available and scalable.

Solution requirements

Table 11-1 shows how the use of WebSphere MQ with a queue sharing group (QSG) meets the major requirements of the project.

Table 11-1 Project requirements

Requirement	Solution
Ability to invoke CICS transactions from multiple channels including Web and ATM	WebSphere MQ provides a simple API that makes it easy for programs running on different platforms to put and get messages from queues.
High availability	Because applications can connect to any queue manager in a QSG, and as all queue managers in a QSG can access shared queues, client applications are not dependent on the availability of a specific queue manager.
High scalability	The use of a QSG enables a scalable solution that takes advantage of the resources across the parallel sysplex. New instances of CICS regions or queue managers can be easily introduced into the QSG as business growth dictates.
Workload balancing	The QSG automatically enables full workload balancing.

11.2 Architecture

The key feature of shared queues is their availability across the sysplex. This enables CICS transactions to run in multiple AORs and access the same queue. Figure 11-1 illustrates the high-level architecture for this project.

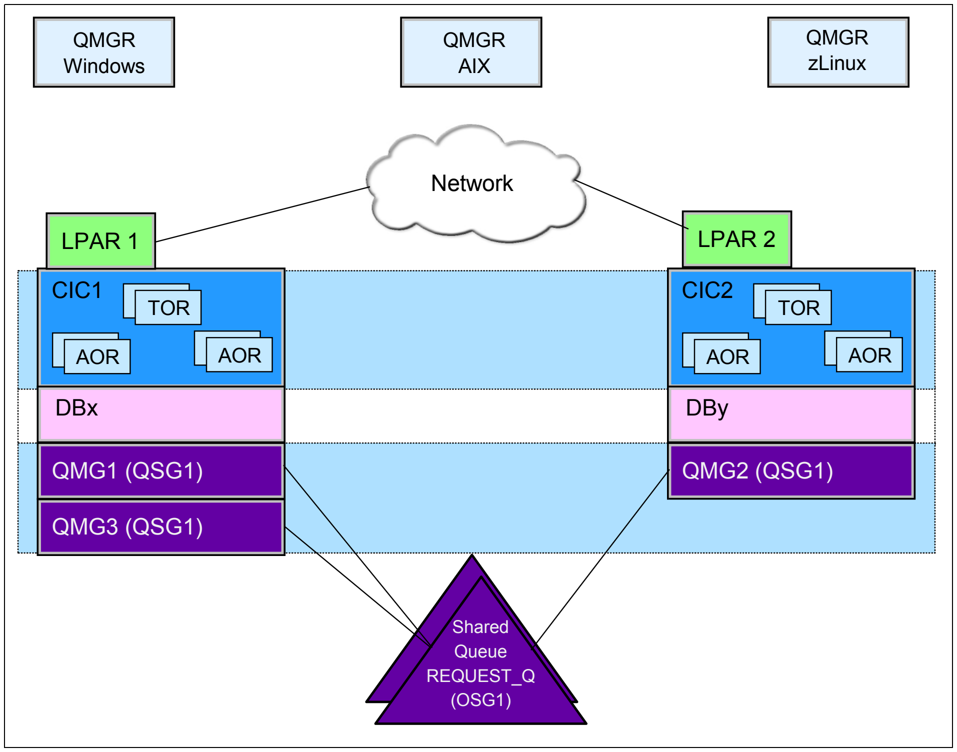


Figure 11-1 Messaging scenario architecture

In Figure 11-1 we can see how multiple instances of the credit card processing application can be running in AORs across the sysplex, all processing messages from the same shared-request queue to maximize throughput.

Furthermore, although applications running outside the sysplex on different servers (that is, Windows, AIX, and Linux) cannot retrieve messages directly from the shared queues (because GET operations can access only those queues that are local to the connected queue manager), they can use PUT operations to write messages to a shared queue.

Typically, one queue manager is configured for each LPAR in the sysplex. However, it is also possible for multiple queue managers to be configured on a single LPAR (QMG1 and QMG3, for example).

11.3 Implementation

The configuration consists of a set of cloned CICS regions spread across two LPARs in a parallel sysplex. The CICS regions share access to a request queue that is held in the coupling facility. Figure 11-2 shows the bank's chosen implementation.

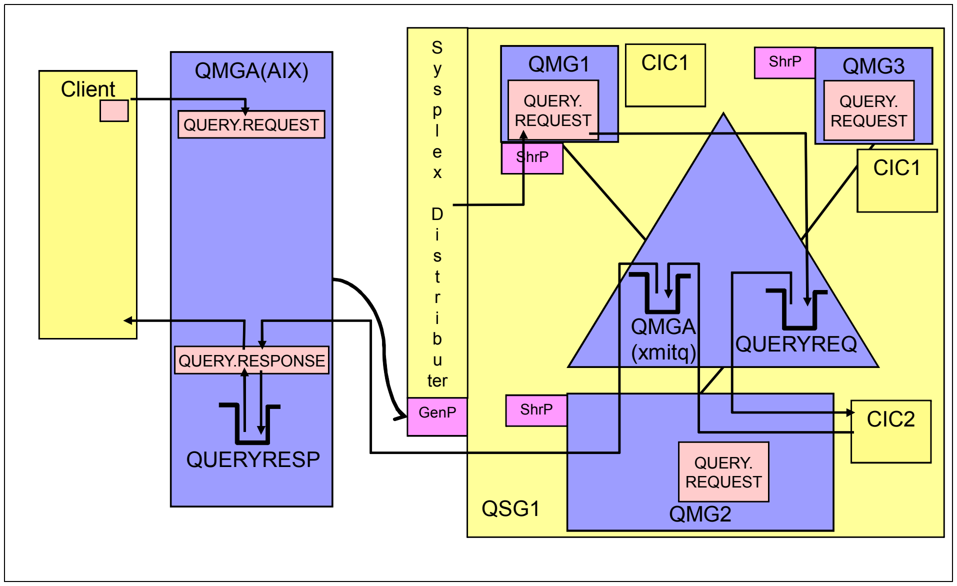


Figure 11-2 Messaging high-availability configuration

The configuration used in Figure 11-2 consists of the following components:

1. The client application connects to the queue manager QMGA running on a distributed system and puts a request message to a queue named QUERY.REQUEST. QMGR is part of a WebSphere MQ cluster.
2. Configure Sysplex distributor (SD) to enable a shared channel. It needs two configurations:
   - Shared port definition in all the three queue managers. All group listeners in the queue-sharing group are defined to be listening on the same port. Define shared port in all three queue mangers (Example 11-1).

Example 11-1 CSQ4INPX member definition

```
In the CSQ4INPX member for the queue manager, define the
following:
START LISTENER TRPTYPE( TCP ) PORT( 31414 ) INDISP(GROUP)
```

– Sysplex distributor configuration in TCPIP parms. Create a profile (Example 11-2).

Example 11-2 Sysplex distributor configuration

```
VIPADEFINE 255.255.255.224 aa.xx.yy.xx
VIPADISTRIBUTE DISTM ROUNDROBIN aa.xx.yy.zz PORT 31414 DESTIP ALL
```

Figure 11-3 shows how the Sysplex distributor is configured.

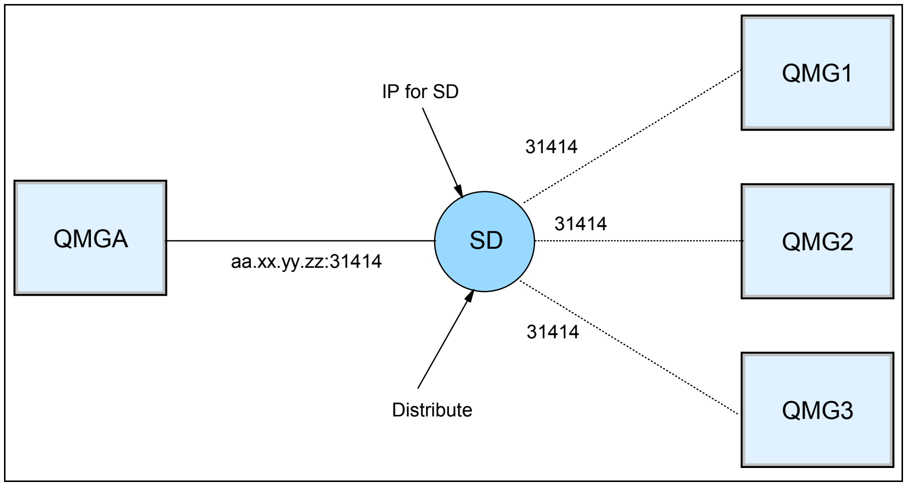


Figure 11-3 Sysplex distributor configuration

3. QMGA connects to a generic port (31414), and a shared channel is started to send the request message to the queue sharing group (QSG1). Sysplex distributor routes the connection request to one of the queue managers (in this example, the connection is established with QMG1).
4. The distributed queue manager then sends a message, which is placed on the shared queue. The queue QUERY.REQUEST is defined as an alias queue (using group definitions so that each queue manager in the QSG has the same definition), which points to QUERYREQ. As this is a local (shared) queue, the request message is placed onto this queue.
5. A trigger is defined for the queue QUERYREQ so that a single CICS trigger monitor transaction is triggered for the message (in this example, CKTI runs on CIC2, which is connected to queue manager QMG2).
6. The CICS application retrieves the trigger message and gets the name of the queue that started the trigger. The started transaction retrieves the request message, and puts the reply message. The ReplyToQ and ReplyToQmgr fields in the request message are specified as QUERY.RESPONSE and QMGA, respectively.

7. The channel for QMGA is a shared channel, so it can be started on any queue manager in the QSG.
8. The response message arrives on QMGA, where the alias queue QUERY.RESPONSE is resolved to QUERYRESP.

11.4 Solution summary

The use of WebSphere MQ with a queue sharing group provides a scalable solution for credit card processing and ensures maximum availability for planned and unplanned queue-manager outages.

Part 4

Appendix

A

Product capabilities

This appendix details which in-service versions of CICS TS and associated products first introduced significant capabilities for each of the integration technologies discussed throughout this book.

Use this information to establish which versions are required for your SOA solution.

Product capabilities

Table A-1 lists the CICS Web services capabilities.

Table A-1 CICS Web services capabilities

Product	Capability
CICS TS V3.1	► SOAP 1.1 and 1.2 ► Web Services Description Language Version 1.1* ► Web Services Atomic Transaction Version 1.0 ► WS-I Basic Profile 1.1* ► CICS Web services asistants ► SOAP Message Security ► Web Services Security: UsernameToken Profile 1.0* ► Web Services Security: X.509 Certificate Token Profile 1.0*
CICS TS V3.2	► Web Services Description Language Version 2.0* ► SOAP 1.1 Binding for MTOM 1.0 ► Simple SOAP Binding Profile 1.0 ► Web Services Trust Language specification in WS-Security* ► WSDL 1.1 Binding Extension for SOAP 1.2
CICS TS V4.1	► Web Services Addressing 1.0* ► WSDL publishing to, and reading from, WebSphere Service Registry and Repository ► Identity Propagation ► Basic authentication for outbound connections ► IPv6
CICS TS V4.2	► Axis2 Java-based SOAP engine, including zAAP offload ► Password phrase support

* Conditionally complies or has restrictions. See the “External standards” topic in the CICS TS information center for further details.

Table A-2 lists the CICS TG for z/OS capabilities.

Table A-2 CICS TG for z/OS capabilities

Product	Capability
CICS TG for z/OS V7.1	► Channel interface support for Java clients ► XA two-phase commit directly into CICS TS via IPIC ► SSL support directly into CICS TS via IPIC
CICS TG for z/OS V7.2	► Server name remapping ► Gateway groups support peer recovery of XA transactions across sysplex

Product	Capability
CICS TG for z/OS V8.0	► Identity propagation with CICS TS V4.2 ► Sysplex-wide highly available Gateway groups ► Channel interface support for the External Call Interface (ECI) v2 clients
CICS TG for z/OS V8.1	► Policy-based dynamic server selection ► ESI over IPIC connections ► ESI for .NET and ECI v2 clients ► Password phrase support ► JCA 1.6 ► Cloud deployment with IBM Workload Deployer ► 64-bit and 32-bit .NET application support ► Channels and containers for .NET and ECI v2 C applications

Table A-3 lists WOLA capabilities.

Table A-3 WOLA capabilities

Product	Capability
WebSphere Application Server for z/OS V7.0.0.4	► Inbound and outbound from CICS TS; inbound to CICS TS limited to one-phase commit ► Thread identity support
WebSphere Application Server for z/OS V7.0.0.12	► Two-phase commit support inbound to CICS TS
WebSphere Application Server for z/OS V8.0.0.1	► JCA connection factory failover ► Round robin request distribution

Table A-4 lists CICS web support features.

Table A-4 CICS web support features

Product	Capability
pre-CICS TS V3.1	► HTTP/1.0 ► WEB API ► Basic authentication ► Secure Sockets Layer
CICS TS V3.1	► HTTP/1.1 including inbound persistent connections, pipelining, chunking* ► Outbound HTTP ► Static content delivery ► Transport Layer Security 1.0 ► Cipher suites and certificate revocation lists
CICS TS V3.2	► WEB APIs enhanced to accept containers ► IP Interconnectivity (IPIC)

Product	Capability
CICS TS V4.1	► Atom Syndication Format ► Atom Publishing Protocol ► Outbound basic authentication ► IPv6
CICS TS V4.2	► HTTP/1.1 pooling of outbound connections ► Limiting the maximum number of persistent HTTP inbound connections ► Atom feeds simplified deployment and administration

*Conditionally complies or has restrictions. See the “External standards” topic in the CICS TS information center for further details.

Table A-5 lists the WebSphere MQ for z/OS capabilities.

Table A-5 WebSphere MQ for z/OS capabilities

Product	Capability
CICS TS V3.2	► CICS-WebSphere MQ attachment integrated with CICS TS ► Threadsafe CICS-WebSphere MQ attachment
CICS TS V4.1	► WebSphere MQ group attach for shared queues
CICS TS V4.2	► Unit of work recovery for shared queues
WebSphere MQ V7.0.0	► Publish and subscribe messaging

Table A-6 lists the CICS sockets capabilities.

Table A-6 CICS sockets capabilities

Product	Capability
Communication Server for z/OS V1R7	► CICS OTE support ► SSL support
Communication Server for z/OS V1R9	► PLT support for CICS immediate shutdown ► Configureable language translation between ASCII and EBCDIC ► Netstat application data support ► Improved error handling and recovery

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *CICS Transaction Server from Start to Finish*, SG24-7952
- *CICS Web Services Workload Management and Availability*, SG24-7144
- *Implementing CICS Web Services*, SG24-7657
- *Application Development for CICS Web Services*, SG24-7126
- *Considerations for CICS Web Services Performance*, SG24-7687
- *Securing CICS Web Services*, SG24-7658
- *High Availability in WebSphere Messaging Solutions*, SG24-7839
- *Smarter Banking with CICS Transaction Server*, SG24-7815
- *Simplifying Integration with IBM WebSphere DataPower Integration Appliance XI50 for zEnterprise*, REDP-4783

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Other publications

This publication is also relevant as a further information source:

- *IBM Techdoc WebSphere z/OS Optimized Local Adapters,* WP101490

Online resources

These websites are also relevant as further information sources:

- A useful definition of SOA is provided by the OSASIS group:

  http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

- For further details on Cloud computing refer to the following website:

  http://www.nist.gov/itl/csd/cloud-020111.cfm

- To read more about CICS TG local mode of operation, refer to this site:

  http://www.redbooks.ibm.com/abstracts/sg247161.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

CICS and SOA: Architecture and Integration Choices

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

CICS and SOA Architecture and Integration Choices

Covers web services, JCA, web support, messaging, and CICS sockets

Is based on CICS Transaction Server V4.2

Includes example integration scenarios

The service-oriented architecture (SOA) style of integration involves breaking an application down into common, repeatable services that can be used by other applications (both internal and external) in an organization, independent of the computing platforms on which the business and its partners rely.

In recent years CICS has added a variety of support for SOA and now provides near seamless connectivity with other IT environments. This IBM Redbooks publication helps IT architects to select, plan, and design solutions that integrate CICS applications as service providers and requesters.

First, we provide an introduction to CICS service enablement and introduce the architectural choices and technologies on which a CICS SOA solution can be based.

We continue with an in-depth analysis of how to meet functional and non-functional requirements in the areas of application interface, security, transactional scope, high availability, and scalability.

Finally, we document three integration scenarios to illustrate how these technologies have been used by customers to build robust CICS integration solutions.

SG24-5466-06 ISBN 0738436739