Architecting Java solutions for CICS

This course presents the compelling reasons for developing Java applications in CICS Transaction Server. The course covers the various usage models of Java applications in CICS, along with an explanation of the technologies involved.

http://www.redbooks.ibm.com/abstracts/crse0301.html?Open

Section 1 Lecture 1

Hello and welcome to the CICS Java course around Architecting Java solutions for CICS.

My name is Ian Burnett and I'm part of the CICS development team, based in the IBM Hursley lab in the UK. This course will describe the reasons for hosting and developing Java applications to run in CICS Transaction Server V5.3. To follow along with this course, you will need to have a background knowledge of mainframe transaction processing, and previous experience of Java application development would be useful.

By the end of this course you will understand the capabilities of CICS as a mixed-language application server and how CICS integrates Java applications into this mixed-language environment. You will also understand the different topologies for deploying Java applications into the CICS mixed language runtime.

Thank you for watching this course.

Section 2 Lecture 1

CICS is a mixed-language application server designed for high-volume online transaction processing. It is a middleware product that runs on IBM mainframes, also known as IBM z Systems. As an application server, it provides services to host applications and access data and messaging engines. These services allow many thousands of users to submit requests and run applications concurrently.

The application server provides full transactional integrity, high security and workload management for the applications. A wide range of industry standard connectors are provided including SOAP and RESTful web services, HTTP, TCP/IP sockets, MQ messaging, APPC, 3270, and other remote clients.

A key value of CICS is the wide variety of programming languages supported. This includes the standard business languages COBOL and PL/I as well as assembler, C, C++, scripting languages such as REXX and PHP and more recently Java. The mixed language support and the simplicity of calling between application components make CICS unique amongst application servers as a true “mixed-language application server”.

So, what are the defining features of mixed-language application support?

The first capability is to allow application components to call other components written in any supported language. This means for instance, that a COBOL component can invoke a Java component, which can then invoke an assembler component. Common CICS interfaces can be used to easily communicate and pass data between these components, and each component can be developed and deployed independently of each other.

The second capability is the provision of a set of standard APIs. Access to CICS managed resources in most languages is provided through the EXEC CICS command based API. This set of APIs are language neutral and the statements are translated at compile time into language specific statements, although some languages such as Java and C++ provide language specific implementations for use at runtime.

The third capability of mixed-language application support is the container managed transactional and security context that is provided for all requests. This is provided by the CICS task and means that a request can run with a common recoverable unit of work and security context, regardless of the number and type of language components involved.

As an example, here we can see an application that consists of multiple components developed in different languages. The initial program is written in COBOL and communicates with the Java and C components using the CICS channel and containers or COMMAREA interfaces. Requests to this application are dispatched within the CICS runtime as a task. The task provides the transactional and security context and manages the data passed between the components.

Let’s look at the first one of these capabilities, namely inter-language calling between components.

Traditionally the interface between CICS application components was based on the communication area or COMMAREA. This requires the data to be structured using language-specific copybooks or structures to describe the location of the elements within the COMMAREA. This structure is then passed to the called component using the CICS link command which is the command mostly widely used to invoke another application component. IBM provides a number of tools to support the mapping between native language structures and Java beans, including the JZOS record generator and the Rational J2C record importer.

More recently the channel interface has become popular as this allows multiple, named data areas known as containers to be passed between programs. Each container can itself be a structured record similar to a COMMAREA, or can be a single field of data. Using channels and containers also has the advantage of built-in data conversion between different encodings for character data, and is not limited to the 32 KB restriction that applies to COMMAREAs. Similar to COMMAREAs, if structured data is passed within a container, then language-specific copybooks can be used to describe the location of the elements, and Java developers can utilize the IBM Java tools to perform the data mappings.

Now let's look at standard APIs.

CICS provides APIs in the form of EXEC CICS commands. These enable CICS-managed resources such as programs, files, or queues to be accessed using a simplified API irrespective of the language being used for development. EXEC API statements, such as the LINK PROGRAM command shown here, are translated at compile time using the CICS translator into language-specific instructions. DB2 and IMS provide the EXEC SQL and EXEC DLI syntax that is also supported for use in CICS applications.

Lastly let's look at the final capability of CICS transaction and security management. In CICS a transaction defines the execution environment for a task. A CICS transaction definition must be installed in the CICS runtime for each different transaction that is invoked and names the first component to be invoked. Multiple requests can be made to the same transaction and each request becomes a unique task. Before the task is executed, the security context is established using the defined CICS security configuration and external security manager such as RACF. Upon execution of a task the CICS runtime creates a unit of work, otherwise known as a global transaction, and this manages the transactional scope when updating recoverable resources. As execution of the task progresses, the transaction and security contexts are propagated across the different application components.

In the next lecture, we'll take a closer look at the Java support in CICS.

Section 2 Lecture 2

So why use Java for developing CICS applications?

Java is a popular programming language and has been adopted by many IT organizations. Many customers are attracted to Java due to the readily available skills, tools and frameworks - enabling applications to be written in a highly productive environment. Support is provided for Java across the industry for many application servers and databases, giving a wide choice of deployment options. Java applications using industry standard APIs can be deployed into the CICS TS V5 mixed-language environment, allowing stepwise modernization of existing applications.

Java application source code is compiled into bytecode, which can then be executed in any Java environment that supports that level of byte code and has the dependent class libraries available. The Java virtual machine or JVM provides the class loaders to load the byte code, an interpreter to execute the byte code, and the just in time compiler or JIT to compile this byte code to hardware specific instructions. The JVM together with the garbage collector provide memory management and further functions are provided to handle threading and synchronization.

Java support on z/OS builds on the Java Standard Edition specification which is supported across all platforms. There are some additional extension libraries on z/OS which provide integration with the z Systems hardware and exploit some of its unique capabilities. Two examples of the z/OS extension libraries are the JCE CCA security provider for use with I/O attached cryptographic accelerators and the JZOS API for access to MVS datasets, consoles and other native functions.

Compared to traditional CICS applications written in other high level languages a key benefit of Java on z/OS is the potential to reduce software licensing costs. This benefit is provided by running Java applications on dedicated specialty processors, including the System z Integrated Information Processor, or zIIP, and historically the System z Application Assist Processor, or zAAP. zIIP specialty processors reduce the cost of Java workloads through a lower cost of acquisition, lower maintenance costs and no IBM software license charges on zIIP capacity. Note that native application code is not eligible for offload to zIIPs.

Let’s now take a closer look at the Java support in CICS. Java support in CICS is provided by the JVM server technology. This is defined in the CSD using a JVMSERVER resource and installed like any other CICS resource definition. The JVM server configuration is stored in a file in a Unix System Services filesystem file termed the JVM profile, which is located using the JVMPROFILEDIR system initialization parameter. This specifies a directory in zFS where JVM profiles are to be found. The JVM profile specifies the options, environment variables and system properties which define how the JVM operates within CICS. The profile is read at startup of the JVM server when the JVM server resource is enabled. Multiple JVM servers are permitted in a single CICS address space and multiple Java applications can be deployed into a JVM server.

There are 3 different types of JVM server that are supported within CICS. The first is the OSGi JVM server, which provides a Java Standard Edition environment. Java applications must be packaged as OSGi bundles and can be dynamically versioned and deployed into the OSGi framework running within the JVM server.

The second is the Liberty JVM server which supports an embedded WebSphere Liberty server for deployment of Web and other Java Enterprise Edition or Java EE applications.

The third is a traditional classpath-based JVM server. This principally supports a variety of hosted environments such as IBM Operational Decision Manager and some embedded CICS functions.

Examples of embedded CICS functions that run within JVM servers include the Axis2 Java SOAP engine, JSON web services, JSON parsers, SAML, and dynamic scripting for PHP. We will only be focusing on the OSGi and Liberty JVM servers in this education series.

OSGi JVM servers support the standard Java SE APIs as well as the JCICS API for accessing CICS resources. In an OSGI JVM server the main method of a Java application acts as the external entry point for other CICS components. This allows CICS programs written in any other language, such as COBOL or assembler, to invoke the Java applications. The LINK command is the principal method used to invoke Java applications, but START commands along with HTTP, 3270 and APPC are also supported. In addition, the OSGi JVM server environment provides integrated support for access to locally connected DB2 databases using either JDBC or SQLJ type 2 connectivity. Access to a local IBM MQ queue manager is also supported using either the JMS2 API or the IBM MQ classes for Java.

OSGi is a Java framework allowing componentization of Java applications. It provides an OSGi runtime, as well as a service registry and support for deployment and packaging. The componentized applications, known as OSGI bundles, can be dynamically added, removed, and versioned within the OSGi runtime. CICS Java programs that target a Java class in an OSGi bundle need to be defined with a JVMCLASS attribute in the program definition. This JVMCLASS attribute references the Java class that is exposed by a CICS-MainClass header in the OSGi bundle manifest.

In the diagram the CICS program definition MYPROG1 is used to define a reference to the class MyClass1. A LINK or START to the program MYPROG1 will invoke the main method in the Java class com.ibm.MyClass1. The Java class will have access to data using a COMMAREA or channel.

WebSphere Liberty is fast, dynamic, easy to use Java EE application server. It is certified for the Java EE 7 full platform, and includes support for Web applications and Enterprise Java Beans. The Liberty server runtime is provided and licensed with CICS and requires no extra installation steps. A JVM server can be configured to run a Liberty server within it. Java EE applications deployed into a Liberty JVM server can access CICS resources such as files and queues through the JCICS API. In addition, applications can access local and remote relational databases using JDBC, and get and put messages from IBM MQ and other supported messaging runtimes using JMS.

Plain Old Java Objects, known as POJOs that are deployed within a web module can be invoked from CICS programs using the EXEC CICS LINK or START command. The entry to the POJOs is defined using a CICS-defined annotation on a specific Java method. This allows Java components deployed into a Liberty JVM server to be part of a broader mixed-language application in CICS.

The Liberty JVM server can be run in one of two modes: integrated and standard mode. Standard mode, provides an environment where all the Java EE 7 full platform features are supported, but with less CICS integration. Java threads running in standard-mode Liberty are not CICS-enabled by default, however, they can use an API to create a new CICS transaction in order to access CICS resources. Standard mode is ideal for hosting Java EE applications that do not require full integration with CICS but would benefit from deployment to System z and higher zIIP offload.

Integrated mode Liberty refers to the fact that the Liberty server is tightly coupled with the CICS runtime, including support for CICS transactions and security, and linking to POJOs in a web application. Integrated mode is ideal for hosting Java EE applications that require frequent access to CICS resources or need to be integrated with other CICS applications.

Java EE applications can be packaged in one of three archive types, WARs, EARs and EBAs. WAR files are used to deploy web applications and supporting libraries. EAR files are used to deploy web

applications and associated EJB components and libraries. EBA files are used to deploy OSGi-enabled web applications and associated bundles. Each of these archive types can be individually packaged or optionally included in a CICS bundle project. Using a CICS bundle provides the ability to deploy multiple application components together with associated CICS resources in one deployable unit.

Now we will take a closer look at the deployment options for these archives. There are three different methods of deploying Java EE applications into a Liberty JVM server, dropins, server.xml application definitions and CICS bundle resource definitions. The simplest option is to use the Liberty dropins function, which is a folder monitored by the Liberty server and is suited for quick deployment into a development environment. Java EE applications can be defined in the Liberty server configuration file, server.xml, using <application> elements. This option allows the application to support customized classloading and qualities of service such as security. CICS bundle resource definitions can also be used install Java EE applications. These are defined in the CSD and allow applications to be managed as part of the wider CICS application infrastructure.

In the next lecture, we will look at the different Java APIs available for use in the CICS environment.

Section 2 Lecture 3

Java has a rich set of APIs that are available to the application developer, and many are supported in a CICS environment. Java Standard Edition or Java SE provides the core of the Java language. Java Enterprise Edition or Java EE extends Java SE and is designed to help developers create large-scale enterprise applications. The JCICS API is provided by CICS and provides access to CICS resources from Java applications, similar to the EXEC CICS command API in other languages. The JDBC and JMS APIs provide access to relational databases such as DB2, and messaging providers such as MQ. The z/OS extension libraries such as IBMJZOS which are provided by the Java runtime environment are also supported.

Java SE is a platform of the Java programming language. The Java SE API consists of several core components such as:

- Base libraries which provide the fundamental function for the Java platform, these are provided by the lang and util packages, and are extended by other base packages providing facilities such as network I/O, security, XML support and the Java Native interface or JNI.
- Integration libraries providing APIs such as JDBC, JNDI and RMI support.
- User Interface toolkits such as AWT, Swing or print services which are principally designed for use in graphical environments.

JCICS provides a simple object hierarchy for accessing CICS resources from Java. Unlike other high level languages, no translation step is necessary during compilation. It provides a Java API that dynamically generates the underlying EXEC CICS calls using the Java Native Interface. The library provided by JCICS includes a range of artefacts that map to the underlying API. Commonly used functions are as follows:

- A Task object to represent the CICS task, which allows units of work to be committed or rolled back, tasks to be abended and other operations to be performed.
- A Program object which provides the ability to LINK to other CICS programs.
- The TSQ and TDQ objects to access CICS temporary storage and transient data queues.
- A set of objects to access VSAM files including KSDS, ESDS and RRDS file types.
- The Start Resource object to start transactions for asynchronous processing.

JCICS commands can throw a CICSConditionException which represents error response codes from the underlying EXEC CICS commands.

Java EE is a platform for hosting enterprise applications and extends the Java SE platform. CICS TS V5.3 supports Java EE 7, along with partial support for the Java EE 6 specification, in a Liberty JVM server.

The Java EE platform comprises a set of Java Specification Requests or JSRs. JSRs are the descriptions of the proposed and final specifications for the Java platform. Each of these JSRs map to the features within the Liberty server, which can be individually enabled.

The web profile is a subset of the Java EE full platform. It allows development of lightweight web-based applications. Java EE 6 introduced the Web Profile, this is a subset of the full platform APIs, capable of addressing the needs of many web applications.

The CICS TS V5.3 Liberty JVM server supports both the Java EE V6 and V7 web profiles. In addition to the core web technologies of JSP and servlet the Java EE 6 web profile supports technologies such as RESTful JSON web services, local EJB support, and context and dependency injection. In Java EE 7 the

Web sockets protocol was introduced into the web profile. Web sockets is an API providing a bidirectional and full-duplex communication channel between web clients and servers.

The Java EE full platform comprehensively defines the Java EE programming model and is designed for the development of large scale enterprise applications. It supports over 30 different specifications. The Java EE full platform can be categorized with the following application tiers:

- The web application tier consists of technology that handles user interaction, and includes technology such as servlets and Java Server Pages.
- The Business tier consists of technology that enables the business logic for an application. This includes enterprise Java beans as well as Java web services including JAX-RS for RESTful web services and JAX-WS for SOAP web services.
- The Enterprise access tier consists of technology to access databases and other enterprise information systems. Examples of this include: Java Database Connectivity or JDBC, which is a low level API for accessing relational databases; Java Persistence API or JPA for persisting data in datastores and mapping to Java objects - it was defined as a replacement for the EJB 2 entity beans specification; Java Messaging Services or JMS for access to messaging providers such as IBM MQ; Java EE Connector architecture or JCA for connecting to enterprise systems; and Java Transaction API or JTA for managing transaction recovery. Java batch was added to the Java EE full platform in version 7 and provides a standard batch programming model in Java.

Development of CICS Java applications is enabled by the CICS Explorer SDK for Java which is an Eclipse plugin that supports all the technologies described in this lecture. The CICS Explorer SDK is used to create CICS bundle projects within an Eclipse workspace. The CICS bundle project can contain one or more of the Java EE application types, and allows them to be deployed as a single entity. An export function is provided to deploy the CICS bundle to zFS, or alternatively the CICS build toolkit can be used to perform an automated build as part of a DevOps process.

In the next lecture, we will look at the common use cases for Java applications deployed into CICS.

Section 2 Lecture 4

In this lecture, we will be looking at a few of the popular use cases for using Java applications in CICS. The first use case centers around Java web applications to support modern web browser interfaces. The second use case is Java web service applications that support JSON and XML based web services directly in Java. The third use case involves writing Java components that provide core business logic for the stepwise modernization of CICS applications. And the last use case is Java batch that provides a solution to process large sets of sequential updates without disrupting online transactions.

Let's look at the detail of the first use case. Web browsers provide sophisticated, modern interfaces on desktops and mobile devices. The server-side components of web applications run in the Liberty JVM server.

Liberty supports the Java EE Web Profile features including Servlets, Java Server Pages, Java Persistence API, and Web Sockets. Together these support the serving of static and dynamic content to web browsers. The dynamic components in the web application can directly access CICS data such as DB2, VSAM, and MQ and invoke CICS programs. It is easy to host web applications from other platforms providing they conform to the Java EE standards. There is the opportunity to simplify administration by consolidating the application tiers – and using the embedded Liberty that is licensed with CICS has the potential to lower costs. The web application can be packaged and managed together with other components such as CICS transaction definitions and policies. Once the application is running in CICS, there is the opportunity to optimize data access using JCICS or the JDBC type 2 connectivity.

The Java web services use case involves JSON and SOAP web service requesters driving APIs exposed within CICS, using the same HTTP protocol and infrastructure as web browsers. Java web services are packaged as web applications and run in a Liberty JVM server. In Liberty, the Java API for RESTful Web Services – or JAX-RS – feature together with the JSON processing feature provide the JSON web services support. For SOAP web services, Liberty provides the features Java API for XML Web Services – or JAX-WS and the Java Architecture for XML Binding – or JAXB. Like web applications, Java web services can directly access CICS data. Java web services supports industry standard formats such as HTTP, JSON, XML, and SOAP. Java is particularly good at handling complex XML. Java web services can be customized using pipeline handlers and filters, for example to perform common security processing and logging. Java on z/OS provides the IBM JZOS data mapping tool to create Java beans that convert Java data to language structures that may be required when calling CICS programs or access data such as that in VSAM.

The third use case involves modernizing CICS applications using new Java components. Each Java component is defined as a CICS program and is then invoked from other CICS programs written in any language using EXEC CICS link or START commands. Data is passed between the components using either a COMMAREA or channels and containers. These Java programs can be packaged as OSGi bundles or as POJOs in a web application. The stepwise modernization of CICS applications by rewriting components in Java and hosting within CICS provides an efficient route to application renewal in CICS. Java development tools and skills can be used to facilitate rapid development. Java components already developed can be reused directly in CICS rather than creating a remote service and relying on the availability of another system. And finally, the CICS mixed language environment is fundamental to this ability through its support for high level languages such as Java, COBOL, or C.

In general batch processing focuses on the non-interactive, background execution of data and compute-intensive work. Java batch is part of the Java EE 7 Full Platform and provides a standard programming model for batch processing. The Liberty JVM server provides the runtime and tools to

use Java batch in CICS. Clients such as enterprise schedulers send requests to start a batch application to the batch dispatcher. The dispatcher uses a job repository and database to understand the steps in the batch application and places work onto a messaging queue. The batch executor takes work from the message queue and manages its execution. The client can inquire on the successful completion of the batch job. Java batch applications can be run with online transactions as resources are committed at regular intervals. This also provides for flexible scheduling options, such as starting a batch at a time of day when there may be spare capacity, and can improve the availability of online transactions. The Java batch scheduler supports parallel processing by running several instances of the application, each of which process a subset of the data. This can lead to better scalability and reduce the completion time. The Java batch framework can automatically restart after failures and skip over bad data. This simplifies batch application development. And lastly, batch applications will become highly optimized for execution due to the iterative nature of batch and long running JVM server environments in CICS.

Section 3 Lecture 1

In this course, we started by looking at mixed-language applications in CICS and how Java, COBOL and other languages can coexist in the CICS runtime.

We then looked in more detail at the Java support within CICS including a deeper dive into the OSGi and Liberty JVM server runtimes. We then reviewed the Java APIs available for use in the CICS environment, including the Java Standard Edition, Java Enterprise Edition and JCICS APIs and how they are supported using the CICS Explorer SDK.

Lastly, we reviewed some common use cases for CICS Java applications. This concludes our CICS Java course - Architecting Java Solutions for CICS.