Comparing Java /COBOL Integration Approaches

With a focus on Java in WAS z/OS or Compute Grid z/OS
Fundamental Issue ...

Given ...

- A WAS z/OS environment (including Compute Grid)
- COBOL Assets
- A desire to integrate the two

Then ...

- What are the integration alternatives available?
- What are the execution runtime requirements and limitations of each alternative?
- What are the pros and cons of each alternative?
Single Picture, Multiple Scenarios, Several Alternatives

WebSphere Application Server (and WCG)

- Java

- COBOL
  - Custom JNI Code
  - IBM COBOL Container

WSGRID

Enterprise Workload Scheduler

CICS

- COBOL
  - CICS Web Services
  - CICS MQ Bridge
  - CICS Transaction Gateway
  - WOLA

- CICS JVM Environment

- Java

- Batch Container

COBOL

- CN11 Feature Pack

See Speaker Notes
Focus of This Document

Offer a quick review of technology
Provide a closer comparison
External "Integration" Using Scheduler

Enterprise Scheduler
TWS, CA-7, Control-M, etc

Submission JCL

WSGRID and MDB Interface

Submission JCL

Compute Grid
Java Batch Programs

Batch
COBOL Module

This is a form of integration
This may be exactly what you need
This is an interesting topic by itself
We'll not cover this in this presentation. If interested, please ask ... happy to discuss.
Batch Container in CICS (CN11 Feature Pack)

Dispatching of jobs is a function of WebSphere Compute Grid

Once inside the CICS batch container the CICS interfaces are available

Program services

JCICS supports the CICS program control commands as described.

<table>
<thead>
<tr>
<th>Methods</th>
<th>JCICS class</th>
<th>EXEC CICS Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>link()</td>
<td>Program</td>
<td>LINK</td>
</tr>
<tr>
<td>SetNextTransaction(), setNextCOMMAREA(), setNextChannel()</td>
<td>TerminalPrincipalFacility</td>
<td>RETURN</td>
</tr>
<tr>
<td>xctl()</td>
<td>Program</td>
<td>XCTL</td>
</tr>
<tr>
<td></td>
<td>Not supported</td>
<td>SUSPEND</td>
</tr>
</tbody>
</table>

From the CICS TS 4.1 InfoCenter

The JCICS class library supports far more than just "Program Services"
CICS Transaction Gateway

1. **EXCI** -- cross memory into CICS region
2. **CTG GW** -- IP to GW, then EXCI into CICS
3. **IPIC** -- IP to CICS listener port

This is limited to WAS → CICS in all cases

**EXCI** limited to 32K COMMAREA; **IPIC** has channel/container support

**Transactional integrity with RRS**
With **EXCI**; **IPIC** uses XA

**Does not require co-location on the same LPAR**

Well-known WAS/CICS integration mechanism with ample documentation to support configuration, uses cases and pros/cons
Custom JNI Code

This forces the bit mode of the WAS server to 31-bit. COBOL is 31-bit and the bit-mode of caller and called for JNI must be the same.

Native method gets passed references to strings and Java objects, which means C Stub code needs to reach back into Java using JNI calls to fetch strings and find objects and their methods.

Exceptions that occur in Java or the native side must be explicitly handled in a coordinated way in both, which is much more involved.

It's possible for native code to hold references to Java objects which then impedes the garbage collection process.

Writing custom JNI code is complicated and creates potential for many problems ... best to avoid this.
High Level Comparison: WOLA and COBOL Container

### WebSphere Optimized Local Adapters
- Cross-memory send and receive transport mechanism
- WAS z/OS and CICS, IMS, Batch, USS, ALCS
- Programming APIs for COBOL, C/C++, PL/I and Assembler
- Bi-directional
- Assert transaction and security in either direction

### WebSphere Compute Grid COBOL Container
- Load COBOL modules into WAS servant region and execute
- Pass and return parameters
- No programming APIs needed; invocation of COBOL module using IBM-written Java-to-Native code
- COBOL may use WAS-managed JDBC Type 2 connection
- COBOL may operate within TX initiated by Java
The Essential Framework of WOLA

1. There's always an external address space to WAS. This is an exchange mechanism between address spaces.

2. External address space must have access to the WOLA modules.

3. Somewhere in the design will be the WOLA APIs, either used directly or indirectly when hidden in CICS Link Server Task.

4. External address space always "registers" into WAS server to establish low-level connection.

5. Then request/response exchange may begin. Either side may initiate, which means WOLA is a bi-directional technology.

6. The Java program in the servant region is shielded from low-level specifics, though it does require use of WOLA RAR or WOLA classes.
The Essential Framework of the COBOL Container

Controller Region

Servant Region Address Space

Java Program
- Create Container
- Create Procedure(s)
- Invoke Procedure(s)

COBOL Container
- COBOL Procedure

Compiled COBOL Modules
LIBPATH or STEPLIB

Note: WAS server must run in 31-bit mode since COBOL is 31-bit

No external address space ... this technology resides inside the servant region address space and executes the COBOL code there

The COBOL container's LE enclave is separate from the address space's LE enclave (clean environment upon creation of COBOL container)

You may create and destroy this COBOL container multiple times within the same servant address space

You may share a JDBC T2 connection between Java and the COBOL program and maintain transactional context because of RRSAF
# Side-by-Side Comparison

<table>
<thead>
<tr>
<th>WOLA</th>
<th>COBOL Container</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High-Level Description</strong></td>
<td>Cross-memory byte array message exchange</td>
</tr>
<tr>
<td><strong>Intended Purpose</strong></td>
<td>Provide a high-speed, low-latency exchange mechanism between WAS z/OS and external address spaces</td>
</tr>
<tr>
<td><strong>Applicability</strong></td>
<td>CICS, IMS, Batch, USS, ALCS; C/C++, COBOL, PL/I, High Level Assembler</td>
</tr>
<tr>
<td><strong>Programming Model</strong></td>
<td>Set of APIs used by external address space to register and exchange</td>
</tr>
<tr>
<td><strong>Security and Transaction</strong></td>
<td>Can propagate TX and identity with limits; see WP101490 for specifics on security and TX support</td>
</tr>
<tr>
<td><strong>Server Bit-Mode</strong></td>
<td>64-bit</td>
</tr>
</tbody>
</table>
Discussion Chart - Use Case Framework

Start

WAS z/OS and COBOL Co-Located?

Yes

Is COBOL presently in CICS?

And intent to leave it there

No

CTG or other loosely coupled technology

WAS z/OS and COBOL Co-Located?

Yes

Is COBOL presently in CICS?

And intent to leave it there

No

CTG or other loosely coupled technology

Direction of invocation?

CICS \textarrow{\rightarrow} \text{WAS}

WOLA

WAS \textarrow{\rightarrow} \text{CICS}

CTG, WOLA or CICS Batch Container

"Standalone" COBOL

Generally speaking, COBOL currently in CICS should remain in CICS.

Access it with WOLA or CTG (or other means, such as CICS batch container) as best suits the application.
Discussion Chart - If COBOL Not in CICS ...

Then you have an option: WOLA or COBOL Container. Which to choose?

Question: which side is the user of service, and which is provider?

From previous chart

Which is seeking service from the other?
- Java seeks COBOL services
- COBOL seeks Java services

Java seeks to use COBOL as a service

This implies an "outbound" model from WAS z/OS

How is COBOL best used, given the nature of the COBOL program itself?
- Batch in Separate Address Space
- As loaded/invoked module

WOLA

COBOL Container

COBOL seeks to use Java as a service

This implies an "inbound" model to WAS z/OS

WOLA has a very powerful set of inbound capabilities:
- Rich API set
- Synchronous/Asynchronous
- Multiple connections