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?

This chart is meant to set the broad context of what is to follow. We have a series of "givens" that form the basis for the discussion.

The first is a WebSphere Application Server for z/OS environment, either by itself or with WebSphere Compute Grid z/OS involved as well. WAS implies Java applications are part of the discussion.

The next "given" is the presence of COBOL assets.

The final "given" is a motivation (or desire) to integrate the COBOL assets with the Java assets into a cohesive process.

With those givens in place, then we may proceed to the questions that come out -- what alternatives are available, what requirements and limitations exist and what are the relative pros and cons of each.

Providing an exhaustive review of every facet of this discussion would be difficult. This document does not set out to do that. Rather, this document sets out to provide a framework of the essential elements of the alternatives so that you may better approach discussions on this topic.
On the surface this appears a very busy chart. It's meant to convey in a single picture the key integration alternatives we will set out to mention or explore in this document.

The following numbered bullets relate to the numbered blocks on the chart:

1. One form of "integration" is done at a level higher than the Java or COBOL code. Enterprise Workload Schedulers (Tivoli Workload Scheduler, or comparable OEM products) may be used to integrate Java and COBOL into a higher-level "process."

2. When both Java and COBOL reside in CICS, the Java program may use the supplied JCICS interface to access program services, and thus integrate with COBOL.

3. The CN11 Feature Pack provides a batch container compatible with WebSphere Compute Grid. Java batch programs in the CICS batch container may then use JCICS to access program services, just as shown in #2.

4. We'll refer to Web Services and MQ as a set of "loosely coupled" integration technologies.

5. The CICS Transaction Gateway product provides several methods by which Java programs in WAS may access COBOL programs in CICS. The chart above is very abstract and does not show the detail of different CTG configurations, such as local EXCI, using the Gateway Daemon, or using IPIC. That detail is described well elsewhere; for the purposes of this document assume all CTG are represented here.

6. WebSphere Optimized Local Adapters (WOLA) is a cross-memory exchange mechanism between WAS z/OS and external address spaces (CICS and/or Batch).

7. This represents the use of custom-written JNI code to access COBOL.

8. This represents the use of the new COBOL Container function of WebSphere Compute Grid.

We will now set out to offer a somewhat high-level comparison of these, with particular focus on a comparison between the use of WOLA (#6) and the COBOL Container (#8).
Focus of This Document

For the next several charts we will offer a review of the technologies highlighted with the orange arrows, and a closer comparison of WOLA and the COBOL Container function.

We won't cover the "loose integration" options involving Web Services or MQ. They are excellent technologies and may do exactly what your requirements suggest. But they are well documented approaches. The focus of this presentation is the "tighter" integration options; specifically WOLA and the COBOL Container.
We start with a review of a higher-level integration using an enterprise scheduler such as Tivoli Workload Scheduler, or one of the OEM products shown. Here the motivation is the integration Java batch processing with traditional batch processing into a higher-level “batch process.”

The chart shows WebSphere Compute Grid but in truth this would work with Java batch using a mechanism such as JZOS. We show Compute Grid here because we wish to make you aware that the product has the capability to integrate with schedulers through a supplied utility called “WSGRID” and an MDB interface to the WCG product.

The WP101783 document at ibm.com/support/techdocs goes into more detail on how this integration technology works.

If this is the level of integration you desire, then please do explore the WSGRID/MDB integration of Compute Grid. We don't go into any more detail on this topic here as our focus it to look at the tighter integration mechanisms, as we mentioned earlier.
Another mechanism for integrating COBOL that's in CICS with Java batch processing is to use the new CN11 Feature Pack of CICS TS 4.1. This feature pack provides a "batch container" environment inside CICS itself that is compatible with Compute Grid and its job dispatching capabilities. The Java program runs in CICS with access to the COBOL through the JCICS interface. In a sense this is just like any Java program using JCICS; we offered this to bring to your attention this capability which is relatively new.
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

On the overview chart earlier we showed the CICS Transaction Gateway function as a single box and a single line, and we noted in the speaker notes how that was an intentional simplification for that chart. Here we'll go into a bit more detail, but still well short of the detail offered by the documentation for the CTG product itself.

CTG is a Java/COBOL integration option if the COBOL resides in CICS, and if the direction of invocation is Java invoking COBOL. CTG is very much a “from outside CICS coming into CICS” technology.

Integration can be fairly tight, using the local connector (which makes use of the EXCI interface of CICS), to more loosely coupled using a network-based protocol (either with the Gateway Daemon or the IPIC function).

CICS Transaction Gateway is a very well known mechanism for integrating WAS with CICS. There is significant documentation on CTG with use cases and positioning advice based on the requirements of your application.
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.

Let's now move to a form of Java/COBOL integration that is far more tightly coupled than those we've reviewed so far. It involves COBOL being invoked from Java using the Java Native Interface (JNI) and your custom JNI code. Generally speaking this is an approach we discourage. It is complicated and there are many potential problems that may arise. This chart summarizes a few.

The new COBOL Container function involves JNI code, but it's IBM-written and supported JNI and its based on considerable experience writing such code.
## High Level Comparison: WOLA and COBOL Container

<table>
<thead>
<tr>
<th>WebSphere Optimized Local Adapters</th>
<th>WebSphere Compute Grid COBOL Container</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cross-memory send and receive transport mechanism</strong></td>
<td><strong>Load COBOL modules into WAS servant region and execute</strong></td>
</tr>
<tr>
<td>WAS z/OS and CICS, IMS, Batch, USS, ALCS</td>
<td>Pass and return parameters</td>
</tr>
<tr>
<td>Programming APIs for COBOL, C/C++, PL/I and Assembler</td>
<td>No programming APIs needed; invocation of COBOL module using IBM-written Java-to-Native code</td>
</tr>
<tr>
<td>Bi-directional</td>
<td>COBOL may use WAS-managed JDBC Type 2 connection</td>
</tr>
<tr>
<td>Assert transaction and security in either direction</td>
<td>COBOL may operate within TX initiated by Java</td>
</tr>
</tbody>
</table>

We'll next drop down and begin to illustrate side-by-side two integration approaches -- one is the WebSphere Optimized Local Adapters (WOLA) and the other is the new COBOL Container function.

WOLA is an address space to address space message exchange mechanism. The COBOL Container is a means by which a COBOL module can be loaded into the WAS z/OS servant address space and a procedure invoked.

WOLA is more flexible in that it supports multiple external address spaces and multiple programming languages. The COBOL Container is for COBOL only and involves no external address space; the COBOL is executed within the address space of the WAS servant region.

The COBOL Container supports the passing and receiving of parameters between Java and COBOL. A supplied utility called the "Call Stub Generator" will generate the stubs and bindings for a given COBOL module.

The two approach the issue of COBOL integration in different ways. WOLA is a message exchange mechanism; the COBOL Container a load-and-invoke mechanism. The two are not mutually exclusive; they may be used in combination as needed.

Let's now turn and perform a quick review of each, then we'll do a more comprehensive side-by-side comparison, then we'll go into a decision flowchart.
WOLA is a low-level message exchange mechanism between a WAS z/OS server and a supported external address space. The external address spaces supported are: Batch, USS, CICS, IMS and ALCS.

As mentioned, this is a very low-level mechanism. To communicate with WAS z/OS, the external address space must have access to the supplied WOLA modules. The external address space must then signal its intent to communicate with WAS z/OS by "registering" into the specific WAS z/OS server. At that point the exchange of messages may commence.

At some level the WOLA programming APIs will be used. Programming APIs exist for the Java side and the native side (COBOL, but also C/C++, PL/I and Assembler). The programming may range from relatively simple to increasingly sophisticated. Much depends on how much control you wish to maintain over the behavior of the exchanges.

The WP101490 Techdoc at ibm.com/support/techdocs is our central repository for WOLA-related documentation. There is considerable detail retained at that Techdoc for understanding and using WOLA.
The COBOL Container function was announced in April of 2011 and will be made available with the WebSphere Compute Grid for z/OS V8 product as well as a fixpack to the WCG 6.1.1 product.

The basis for this function is the ability to load a specified COBOL module into the address space of the WAS z/OS address space and invoke the named procedure or procedures. This is not a message exchange mechanism like WOLA; this is a direct load-and-invoke mechanism.

At a very high level this is doing what custom JNI code would do. The difference is the JNI code in use for this is written and supported by IBM rather than by you.

Further, this function allows you to pass parameters into the COBOL procedure and retrieve the results. A side utility called the “Call Stub Generator” is supplied to generate the call stubs and data bindings for your Java program.

JDBC Type 2 connections created in WAS can be shared with the COBOL procedure. This is limited to T2 because by using RRSAF the transactional context created in Java can be propagated into the COBOL program and transactional integrity maintained.
Here's a side-by-side comparison of WOLA and the new COBOL Container function.

As you can see, the two take different approaches to the question of integrating Java and COBOL.

<table>
<thead>
<tr>
<th>Side-by-Side Comparison</th>
<th>WOLA</th>
<th>COBOL Container</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-Level Description</td>
<td>Cross-memory byte array message exchange</td>
<td>COBOL module load and invoke</td>
</tr>
<tr>
<td>Intended Purpose</td>
<td>Provide a high-speed, low-latency exchange mechanism between WAS z/OS and external address spaces</td>
<td>Provide a means of utilizing COBOL assets within the context of a Java batch application in Compute Grid</td>
</tr>
<tr>
<td>Applicability</td>
<td>CICS, IMS, Batch, USS, ALCS; C/C++, COBOL, PL/I, High Level Assembler</td>
<td>COBOL invoked within servant region address space of WCG endpoint</td>
</tr>
<tr>
<td>Programming Model</td>
<td>Set of APIs used by external address space to register and exchange</td>
<td>Load of compiled module with direct invocation</td>
</tr>
<tr>
<td>Security and Transaction</td>
<td>Can propagate TX and identity with limits; see WP101490 for specifics on security and TX support</td>
<td>COBOL run under WAS thread identity; shared JDBC T2 may participate in global transaction initiated by Java Batch program</td>
</tr>
<tr>
<td>Server Bit-Mode</td>
<td>64-bit</td>
<td>31-bit</td>
</tr>
</tbody>
</table>
We will now take you into a discussion/decision flow chart to help position the various alternatives in your mind. Please do not consider this a fully deterministic flowchart ... there are variations and considerations you may have in mind that alter the final decision. Take this as a discussion framework.

The first decision point illustrated is concerned with whether WAS and COBOL is co-located on the same LPAR. If they are not, then by definition the technologies that require co-location are removed from consideration. Then the "loosely coupled" alternatives such as network-based CTG, or web services or MQ would be more appropriate.

But if they are on the same LPAR, then the next question is whether COBOL is presently in CICS, and if the desire is to maintain the COBOL asset in CICS. If the answer is "no" ... then we move to the next chart. But if the answer is "yes" then we get to the next question, which involves the direction of invocation.

We ask this because only WOLA supports the CICS-into-WAS invocation direction, while the more traditional WAS-into-CICS flow is supported by WOLA as well as CTG or the CN11 CICS Batch Container mechanism.

Our general point here is that COBOL assets currently in CICS and doing a good job in CICS should be left there. There are variations on this ... for example, you could choose to leave COBOL in CICS and make use of the COBOL Container function for a specific COBOL procedure you wish to invoke inside the WAS z/OS servant.

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

COBOL seeks to use Java as a service
This implies an "inbound" model to 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

WOLA

On the previous page we determined that the COBOL asset was not in CICS. That means things like CTG are off the table.

The next question we pose has to do with the nature of the relationship between Java and COBOL; specifically, which will serve the role as provider of a service to the other. The answer to this question tends to suggest a direction of invocation -- inbound to WAS or outbound from WAS -- and that tends to suggest certain alternatives.

So if Java is seeking to use COBOL as a service, it suggests an "outbound" model from WAS. Then we must consider how that COBOL asset is best used -- as a batch program in its own address space? Or as a module loaded and invoked? WOLA addresses the issue of a batch process in a separate address space; the new COBOL container addresses the other.

If the answer to the earlier question is that COBOL is seeking to use Java as a service, then that suggests an "inbound" model. Here WOLA is indicated as it provides an inbound model and provides a rich set of functionality to accomplish that.

Again ... variations on these recommendations exist, and by no means are we suggesting this is the definitive decision tree.

End of Document