CICS and Java

OSGi within business critical mainframe environments

Java and OSGi Developer Edition
Big Picture: Where to go with the CICS Development

Today (sample scenario)

- Cobol-Modules
- New functions
- DB2
- System z

Modernisation

- How to implement?
- How to use services of other systems?
- Hard to maintain/change?

Target Architecture?

Java Modul

App Server

External Systems
A Java Anology
Agenda

- Mainframe: What it is, what it is for and why should Java be used on it
- OSGI – A Service Oriented Approach for Java
- OSGI and CICS: How does the JVMSERVER change the way of Java in CICS
- Plattform independent development
- Conclusion and Outlook: Java, CICS and Cloudstyle deployment
CICS and Java

Agenda

- Mainframe: What it is, what it is for and why should Java be used on it
- OSGI – A Service Oriented Approach for Java
- OSGI and CICS: How does the JVMSERVER change the way of Java in CICS
- Plattform independent development
- Conclusion and Outlook: Java, CICS and Cloudstyle deployment
IBM System z

- System z is the flagship mainframe operating system
- Mainframe can simultaneously run many OSs including Linux
- Facts about the mainframe:
  - 2 power supply units (PSUs)
  - Used by top 50+ banks in the world
  - Record breaking 9445 transactions/sec
- Specialty engines
  - zAAP – application assist processor (Java)
  - zIIP – integrated information processor (databases)
  - Cell processor - virtual worlds
Batch vs. Online

**Batch job**
- Input Data
- Application program
  - Process data to perform a particular task
- Output Data

**Online transaction**
- Query
- Reply
- Application program
  - Accesses shared Data on behalf of an online user
Business Application Requirements

Business Logic

Application

Naming
Locking
Administration
Exception Handling
Recovery
Performance
End-to-End Integrity

CICS
Application Server

Multithreading
Logging
Scheduling
Security
Connectivity
Queue Management
Time Control

Communications

Database Manager

Operating System

Hardware
CICS – An Application Server

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>3270</th>
<th>CTG</th>
<th>Web</th>
<th>Web services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Services</td>
<td>Transactions</td>
<td>WLM</td>
<td>Monitoring</td>
<td>Statistics</td>
</tr>
<tr>
<td>Programs</td>
<td>Assembler</td>
<td>COBOL</td>
<td>PL/I</td>
<td>C</td>
</tr>
</tbody>
</table>

CICS TS runs on System z…
Linking CICS Programs

Data is passed using a Channel or COMMAREA
Transactions & Tasks

Resource Definitions

EGUI       Program DFH0XGUI
ECFG       Program DFH0XCFG

Task 1
Task 2
Task 3
Transaction Flow
VTAM or TCPIP receives message & passes to protocol handler

Protocol handler copies message in buffer for later processing
Task Control

- Check if input transaction ID valid
- Create storage area called task control area (TCA) for task representation
Program Control

- Locate and invoke first program for transaction
- Program defined as PROGRAM resource in CICS and referenced by transaction
- Load program from library if not already in memory
User Application Program

Example

- Receive message
- Perform some logic
- Send message
- Return to CICS
Message Input

- Transform received input data if necessary
File Access

- VSAM file must be defined in FILE definition in CICS
- Perform create/ read/ update/ delete
- CICS provides completion information
External Resource Manager

- Extension of standard CICS resource set by External Resource Manager (ERM) like DB2 or WebSphere MQ or DBCTL (IMS-DB)
- Log stream (MVS Logger) used to store recovery information
Queues

- Temp to store information for later
- Transient to pass data to other CICS task or batch job as sequential file
- CICS uses TD (transient data) Queues for message logging
Trace

- Summary of all activities during execution
- Trace written into trace table or sequential trace file
Write dump if serious error or abnormal termination
Message Output

- Send back reply to user
Ending the Transaction
Java is just another Language which is running on the Mainframe
Java on Mainframes - what is different and why here?

- IBM uses its own implementation of a JVM on mainframes that uses the underlying platform architecture.
- The Java workload can be offloaded to a zAAP processor.
- The Java logic can be a bridge between the mainframe and the distributed world.
- Java is a common language on many platforms that can help to find a dialog between the departments.
- Java is a language that is well known by many new professionals and so a good common ground, when they start to develop for mainframe applications.
- The language encourages a good design and loose coupling of components.
70+ new instructions used by Java

- Register high-word facility
  - Facilitates use of upper 32-bits of registers
- Interlock facility update
  - Better Java concurrency
- Non-destructive operands
  - Reduce path-length
- Conditional-load/store
  - Remove expensive branches
- Instruction scheduler for Out-of-Order pipeline

Hardware for Java

- New Out-Of-Order pipeline design
- New larger cache structure
- Higher clock speed (~5.2GHz)
z Hardware: Java as a Workload Optimized System

- Continued aggressive investment in Java on zEC12
- Significant set of new hardware features tailored for and co-designed with Java
  - **Hardware Transaction Memory (HTM)**
    - Better concurrency for multi-threaded applications
  - **Run-time Instrumentation (RI)**
    - Real-time feedback on dynamic program characteristics
    - Enables increased optimization by JRE
  - **2GB page frames**
    - Improved performance targeting 64-bit heaps
  - **Page-able 1MB large pages using flash**
    - Better versatility of managing memory
  - **New software hints/directives**
    - Data usage intent improves cache management
    - Branch pre-load improves branch prediction effectiveness
  - **New trap instructions**
    - Reduce over-head of implicit bounds/null checks
Java packaging and class hierarchies

- Java modularity:
  - Classes contain data and logic
  - Packages contain these classes and organize them
    - This is just a virtual form of organization
  - Jar files contain the classes and are the base for enterprise applications

- Java visibility settings:
  - Private, protected, package private, public

- At the runtime, there are just a lot of classes on a classpath

There are some features missing: jar visibility, versioning, dependencies
The Classpath – What was called a load lib concatenation before
Dependencies between different Classes

If you need a new Version of this library, this change always effects all referencing classes in the JVM!
Mainframe: What it is, what it is for and why should Java be used on it

OSGI – A Service Oriented Approach for Java

OSGI and CICS: How does the JVMSERVER change the way of Java in CICS

Plattform independent development

Conclusion and Outlook: Java, CICS and Cloudstyle deployment
OSGi – A dynamic module System for Java

It provides a general-purpose, secure, and managed Java framework that supports the deployment of extensible and downloadable applications known as bundles.

The OSGi Alliance, Core Specification

Sounds familiar? Isn't that already possible with Java?
What is in OSGi for you?

- The OSGi Service Platform specifies a modular architecture for dynamic component based systems
  - Execution Environment
  - Module Layer
  - Life Cycle Layer
  - Service Layer
  - Security-Layer (optional)
- Runs on a variety of standard Java profiles.
- Introduces Bundles as modules
Each bundle has its own class loader!

- **Bundle C**
  - **Import-Package**: package.a
  - **Export-Package**: package.c

- **Bundle A**
  - **Export-Package**: package.a

- **Bundle B**
  - **Export-Package**: package.b

- **Export-Package**: package.a
  - **Import-Package**: package.a

  - **Export-Package**: package.b
OSGi bundles

OSGi Bundle – A jar containing:

- Classes and resources.
- OSGi Bundle manifest
  - Bundle-Version: Multiple versions of bundles can live concurrently.
  - Import-Package: What packages from other bundles does this bundle depend upon?
  - Export-Package: What packages from this bundle are visible and reusable outside of the bundle?

Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Hello Plug-in
Bundle-SymbolicName: com.ibm.cics.server.examples.hello
Bundle-Version: 1.0.0
Bundle-RequiredExecutionEnvironment: J2SE-1.4, J2SE-1.5, JavaSE-1.6
Import-Package: com.ibm.cics.server;version="[0.0.0,2.0.0)"
Export-Package: examples.hello
When bundles are installed into OSGi framework, the module layer:

1. Processes metadata in the manifest file
2. Reconciles declared external dependencies against exports & version information declared by other installed modules
3. Works out all dependencies & calculates independent required class path for each bundle

Ensure that:

- Each bundle provides **visibility** only to Java packages that it explicitly **exports** - exported at specific versions possible
- Each bundle declares its package **dependencies** explicitly - import at specific / range of versions possible
- Multiple versions of a package can be available concurrently to different clients
Old World and New World
Bundle-Lifecycle

- Installed
  - install
  - refresh update
- Resolved
  - resolve
- Uninstalled
  - uninstall
- Starting
  - start
  - Policy: eager/lazy
- Active
  - stop
- Stopping
  - uninstall

Actions:
- install
- resolve
- refresh update
- start
- stop
- uninstall

States:
- Installed
- Resolved
- Uninstalled
- Starting
- Active
- Stopping
OSGI Enterprise Edition

- Extends the OSGi core specification with important features that are used within enterprises
- Enabled OSGi for enterprise applications
- Main focus is set on the existing JEE platforms
- Within this enterprise extension, there are two methods to declare services
  - Declarative Services (DS)
  - Blueprint
- The reference implementation is Apache Aries
  - Is based on eclipse equinox as underlying OSGi framework
  - The implementation of the OSGi EE standard itself is done within OSGi packages that are running within the equinox framework
Small Quiz

- **Small Quiz:**
  - Why is the JCICS package numbered with: 1.0.0, 1.100.0, 1.201.0 and 1.300.0?
Some best practices, when developing object oriented with OSGi

- Always separate interface and implementation, design an API:
  - An API is the data exchange point within an application and is not just a Java Interface definition. It contains
    - Objects that are exchanged
    - Methods that are used
    - ...
  - Use a loose coupling between the OSGi bundles and always use/import the API (consequently never export any packages from the implementation bundles)

- Use a declarative based service approach within applications:
  - Avoid the programming of service registration

- Package 3\textsuperscript{rd} party libraries also within OSGi bundles, to include them into the lifecycle

- Use a semantic versioning to identify the bundles
  - First high level qualifier change is not compatible with earlier versions (API Change)
  - Second high level qualifier changes are backward compatible, but need the implementation to be changed on the provider side, to provide the new functionality for new clients
  - Third high level qualifier changes show fixes, that are transparent to users and API
Agenda

Mainframe: What it is, what it is for and why should Java be used on it

OSGI – A Service Oriented Approach for Java

OSGI and CICS: How does the JVMSERVER change the way of Java in CICS

Plattform independent development

Conclusion and Outlook: Java, CICS and Cloudstyle deployment
Flashback: CICS Pooled Server Architecture
Flashback II: Old pooled Architecture

CICS

JVM  JVM  JVM  JVM  JVM

RDO

Transaktion  Program

SIT

zFS

USS Directory with Class Files

Profile
The JVM Server
- CICS requests storage from MVS, sets up a Language Environment enclave, and launches the 64-bit JVM in the enclave.
- IBM® 64-bit SDK for z/OS, Java Technology Edition, Version 6.0.1
- Up to 256 parallel tasks/JVM & 1024/CICS
- Applications
  - Must be threadsafe
  - deployed as OSGi bundles (in CICS bundles)
- Dynamic updates without restart
- No EJB support
New JVM-Server Architecture
How all the XML manifests belong together

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<manifest xmlns="http://www.ibm.com/xmlns/prod/cics/bundle"
bundleVersion="1" bundleRelease="0" build="Not Found">
    <meta_directives>
    </meta_directives>
    <define name="hello" type="http://www.ibm.com/xmlns/prod/cics/bundle/OSGIBUNDLE"
path="hello.osgibundle"/>
</manifest>
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<osgibundle
symbolicname="com.ibm.cics.server.examples.hello"
version="1.0.0" jvmserver="DFH$JVMS"/>
```

Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Hello Plug-in
Bundle-SymbolicName: com.ibm.cics.server.examples.hello
Bundle-Version: 1.0.0
Bundle-RequiredExecutionEnvironment: J2SE-1.4,J2SE-1.5,JavaSE-1.6
Import-Package: com.ibm.cics.server;version="[0.0.0,2.0.0)"
CICS-MainClass: examples.hello.HelloCICSWorld,
examples.hello.HelloWorld
Most Parts of the descriptor are the same, except the CICS-MainClass:

- CICS needs to know, which Class can be called as a program
- CICS processes the metafile before it is handed to the OSGi framework and the information is stored in the CICS repositories.
- This information can be discovered using CICS Explorer, but not CEMT!

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Hello Plug-in
Bundle-SymbolicName: com.ibm.cics.server.examples.hello
Bundle-Version: 1.0.0
Bundle-RequiredExecutionEnvironment: J2SE-1.4, J2SE-1.5, JavaSE-1.6
Import-Package: com.ibm.cics.server;version="[0.0.0,2.0.0)"
Export-Package: examples.hello
CICS-MainClass: examples.hello.HelloCICSWorld, examples.hello.HelloWorld
```
How are the Java Resources Managed within CICS and zOS in general

- With Java CICS leaves once again the traditional way of definitions within the CSD, like it does with the Event Bindings.
- The reasons are manifold, but the biggest impact comes from the increased complex artifacts
- So how are bundles managed?
  - Like the resources in the distributed world within zFS files:
Mainframe: What it is, what it is for and why should Java be used on it

OSGI – A Service Oriented Approach for Java

OSGI and CICS: How does the JVMSERVER change the way of Java in CICS

Platform independent development

Conclusion and Outlook: Java, CICS and Cloudstyle deployment
Which tools did we discovered yet?

- Java:
  - Language that is able to run on the host
  - Object oriented approach

- OSGi:
  - Service Oriented Approach
  - Exchange of modules that are in Service
  - Available in nearly all Java environments

- So what is missing?
  » How is data accessed?
  » How are connections managed?
  » How is the output formatted?
  » How can other programs be integrated?

- The following approach was developed by our service department led by Philipp Breitbach
The application needs to be designed within a Framework

- The framework has to encapsulate the platform specific interfaces:
  - Database Access
  - Connection Management
  - Program calling
  which are provided by:
  - JEE
  - CICS
  - Batch
- The business logic itself resides in plain Java objects (POJO)
- The framework needs to define specific a specific interface, that matches the requirements of the applications
- The definition of that framework is sometimes already done by taking the JEE interface and design a testing framework for workstations
Online Architecture (CICS)

CICS

Online Controller  TxMgr  Sessioning  LicenseMgr  TransportMgr

Java Framework

Service Control Layer

Business Service Layer

Data Access Layer

ConfMgr  LogMgr  CallMgr  ErrorMgr  ConnMgr

COBOL Logic

CICS LINK
Batch Architecture

Batch Address Space

JVM

Batch Container

Checkpoint Algorithm

Batch Job Step

Business Service

Data Service

BatchData Stream

Java Framework

COBOL

JNI
Batch Architecture in more detail

- Batch Container initializes BatchDataStream (BDS) and Batch Job Step (BJS)
  - BDS can open cursors, files, … on init
- Batch Container loops over BJS
  - BJS reads next input data from BDS on each iteration and calls one or more Business Services to do its Batch logic
- Batch Container issues checkpoints (e.g. all 100 iterations, all 1000 rows written, …) depending on a Checkpoint Algorithm (CA)
  - BJS feeds CA with necessary data
  - BJS has a pre-checkpoint hook to facilitate e.g. JDBC Batch Updates
Online and Batch Architecture together

CICS – Online

- Online Controller (Java)
- Service Control

JES2 – Batch

- Batch Container
- Batch Job Step
- CA
- BDS

Business Service

Data Service
Agenda

- Mainframe: What it is, what it is for and why should Java be used on it
- OSGI – A Service Oriented Approach for Java
- OSGI and CICS: How does the JVMSERVER change the way of Java in CICS
- Plattform independent development
- Conclusion and Outlook: Java, CICS and Cloudstyle deployment
Big Picture: Where to go with the CICS Development

Today (sample scenario)

- New functions
- Cobol-Modules
- How to implement?
- Hard to maintain/change?
- DB2
- System z

Modernisation

- Standard interfaces
- How to use services of other systems?

Possible Target Scenario

- New functions realised with Java
- Efficient programming model
- Reduced maintenance costs
- Stepwise integration
- Java Modules
- Cobol-Modules
- DB2
- System z
- App Server
- External Systems
Beispiel: Nutzung eines internen Web Services

User

AIX
Web Anwendung
SOAP/HTTP
AIX
Web Service

zOS
CICS
Business Transaction
CTG
Secure Data

SOAP/HTTP
JDBC
Verbesserung durch doppel-Deployment

User

AIX
- Web Anwendung
  - SOAP/HTTP

AIX
- Web Service

zOS
- CICS
  - Business Transaction
  - Link
  - Web Service
  - JDBC

CTG

Secure Data

JDBC

Web Service

CICS Link
Die entstandene Verbesserung lässt sich messen

- Performance Verbesserung durch den lokalen DB2 Zugriff
- Auf Seiten des Hosts wesentlich einfacher zu warten
- Die Verfügbarkeit wurde durch Plattform Konsolidierung erhöht
Running Liberty within a JVM Server brings Servlets in CICS with just a few more Options within the profile file.
How does Liberty within CICS work?
Liberty Code was NOT changed, just extended via extension Points.

Liberty – Web Feature
(CICS)ExecutorService.execute (Runnable)
Worker thread

JVM
Liberty – Web Feature
HttpListener

CICS

LE enclave

JDBC
Link to COBOL
Etc.

Same Task Context
Why implement it like that?
- Benefits of Hybrid Threads

- Each 'Invocation' (think Servlet Request) on a Hybrid Thread is also a CICS Transaction (Has a Tranid, Task Context etc).
- This gives you
  » A single common Transaction (UOW) and CICS Managed JDBC
  » Which can cross between Java and Cobol
  » Full JCICS API Access
  » In particular, LINK and access to VSAM
  » WLM (CICS WLM, Performance Classes etc).
  » Monitoring / Statistics
  » CICS Transaction Tracking / Association Data
"Hey Simon, I need to test my new version of the payroll *application*."

"Sure Abigail, you should get the latest payroll test *platform* from the repository, I'll send you a link."

"Thanks, I'll *deploy my app* onto that platform on the development plex later today."

"That should be fine, just check with Oliver that the *policies* on the plex are going to be OK for your app's changes."
First-class applications

A collection of one or more CICS bundles
Life-cycle as a single entity
Measure and control resource usage
Develop in Eclipse/Rational
Share and promote through SCM

Application Lifecycle
- INSTALL
- ENABLE
- DISABLE
- DISCARD

Application
- Name: org.maw.banking.Loans
- Version: 1.2.1
- Resources: LIBRARY, PROGRAM, TRANSACTION, URIMAP
- Dependencies: DB2CONNECTION, JVMSERVER, TCPIPSERVICE, ...
- Entry points:
  - operation: browse, update, ...
  - resource: PROGRAM
First-class platforms

Set of one or more region types
Life-cycle as a single entity
Hides complexity of underlying topology
Provides services for Applications

*Control Applications through Policy*
If we have time: What does a programmer in a new language first?
The “Hello World” IVT :)
Questions?

Tobias Leicher
Client Technical Professional
– CICS & CICS Tools
IBM-Allee 1
D-71139 Ehningen
Mobil: 0151 – 15 16 24 89
Mail: tobias.leicher@de.ibm.com

I ❤️ CICS

Please provide Feedback :)