

Java auf z/OS – eine Alternative die sich lohnt.

Rodney Krick

IT Senior Consultant

aformatik Training & Consulting GmbH & Co KG

rk@aformatik.com

www.aformatik.com

Executive Summary

Viele Unternehmen stecken in einer Zwickmühle: ein großer Teil der so genannten „Assets“ läuft weiterhin mit COBOL, PL/I und Assembler Programmen auf z/OS. Das Wissen um die Pflege und Weiterentwicklung dieser Prozesse wird jedoch rar. Neue Mitarbeiter beherrschen die etablierten 3GL Sprachen oftmals nicht. Java und die LAMP-Welt (Linux, Apache, MySQL, PHP) haben die Ausbildungsstätten erobert. Unternehmen die den Mainframe im Einsatz haben wollen diese gut ausgebildeten Fachkräfte für sich gewinnen, aber gleichzeitig auf die hohe Verfügbarkeit von Mainframes nicht verzichten. Die Frage die sich stellt ist: kann man dieses neue Wissen mit der hoch verfügbaren z/OS Plattform in Einklang bringen und welchen Mehrwert bringt dies? In diesem WhitePaper wird ein Szenarium beschreiben, bei welchem nicht nur die Zwickmühle aufgelöst wird, sondern auch noch zusätzlich ein Mehrwert in Form einer Einsparung von ca. 50% geschaffen wird.

Die Voraussetzungen

Wissen ist Macht! Oder, wie von Francis Bacon ausformuliert: „Wissen und Macht des Menschen fallen zusammen, weil Unkenntnis der Ursache die Wirkung verfehlen lässt“. Auf gut Deutsch: nur diejenigen die wissen, was sie wollen und womit sie die Ziele erreichen können, werden am Ende erfolgreich sein. Auf unser Thema angewendet bedeutet das u.a.:

- was bietet uns die z/OS Umgebung an, im Bezug auf die Java Technologie?
- welche Tools brauchen wir zusätzlich, um eine effiziente Laufzeitumgebung zu konfigurieren?
- welches Wissen wird benötigt, um eine komplette Lösung zu implementieren?
- ist Java eine sinnvolle Alternative zu den bewährten 3GL Sprachen?

Ein Proof-of-Concept (POC)

Um das Ganze nicht allzu theoretisch zu formulieren, soll der Einstieg mit Hilfe eines Beispiels erläutert werden. Ein Kunde von aformatik wird im Bereich Zollverfahren unterstützt, d.h. wir beraten und helfen bei der Anpassung, Weiterentwicklung und Automatisierung von Zollprozessen (besonders bei den Zollverfahren von wirtschaftlicher Bedeutung). Die Anwendungen, die in den letzten Jahren realisiert wurden, verwenden folgende Architektur:

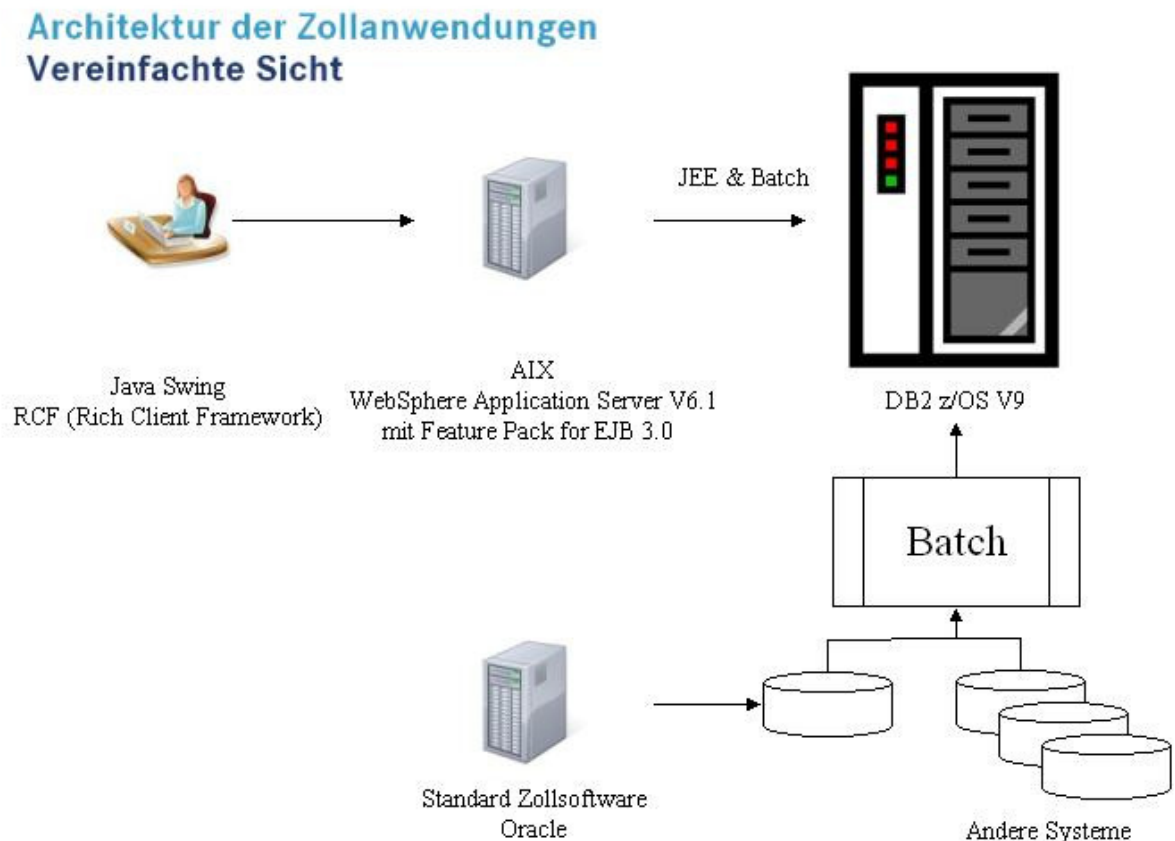


Abbildung 1: Architektur der Zollanwendungen

Hier wird nur eine kurze und sehr vereinfachte Zusammenfassung der Architektur vorgestellt.

- Der Client ist in Java Swing implementiert und greift auf die JEE Komponenten über einem vom Kunden entwickelten Framework (im Bild nicht dargestellt).
- Die Anwendungslogik hat zwei Kernkomponenten: ein JEE Teil auf WebSphere Application Server (WAS) und ein Batch Teil auf z/OS.
- Es sind Batchabläufe sowohl auf der AIX Seite als auch auf der z/OS Seite, muss Massenverarbeitung von Daten.

Das Team, welches diese Applikationen betreut, setzt sich aus Spezialisten aller IT-Bereiche zusammen, wobei von allen erwartet wird, dass ein Grundwissen über alle Aspekte vorhanden ist. D.h. einige beschäftigen sich intensiver mit den Batch-Abläufen, kennen aber auch die JEE Schicht (EJB 3.0, Swing, IBM Rational Application Developer, SVN, usw). Andere betreuen die JEE bzw. Client Applikation (die Java Welt), verstehen aber auch die Batch Implementierung (JCL, COBOL, ISPF, usw).

Die Vision bei der Betreuung und Weiterentwicklung der Anwendungen ist die komplette Automatisierung aller Prozesse und das setzt voraus, dass die zwei Hauptkomponenten (JEE und Batch) miteinander Informationen austauschen. Genau hier wurde bei diesem POC angesetzt.

Folgender Prozess wurde als Kandidat auserwählt:

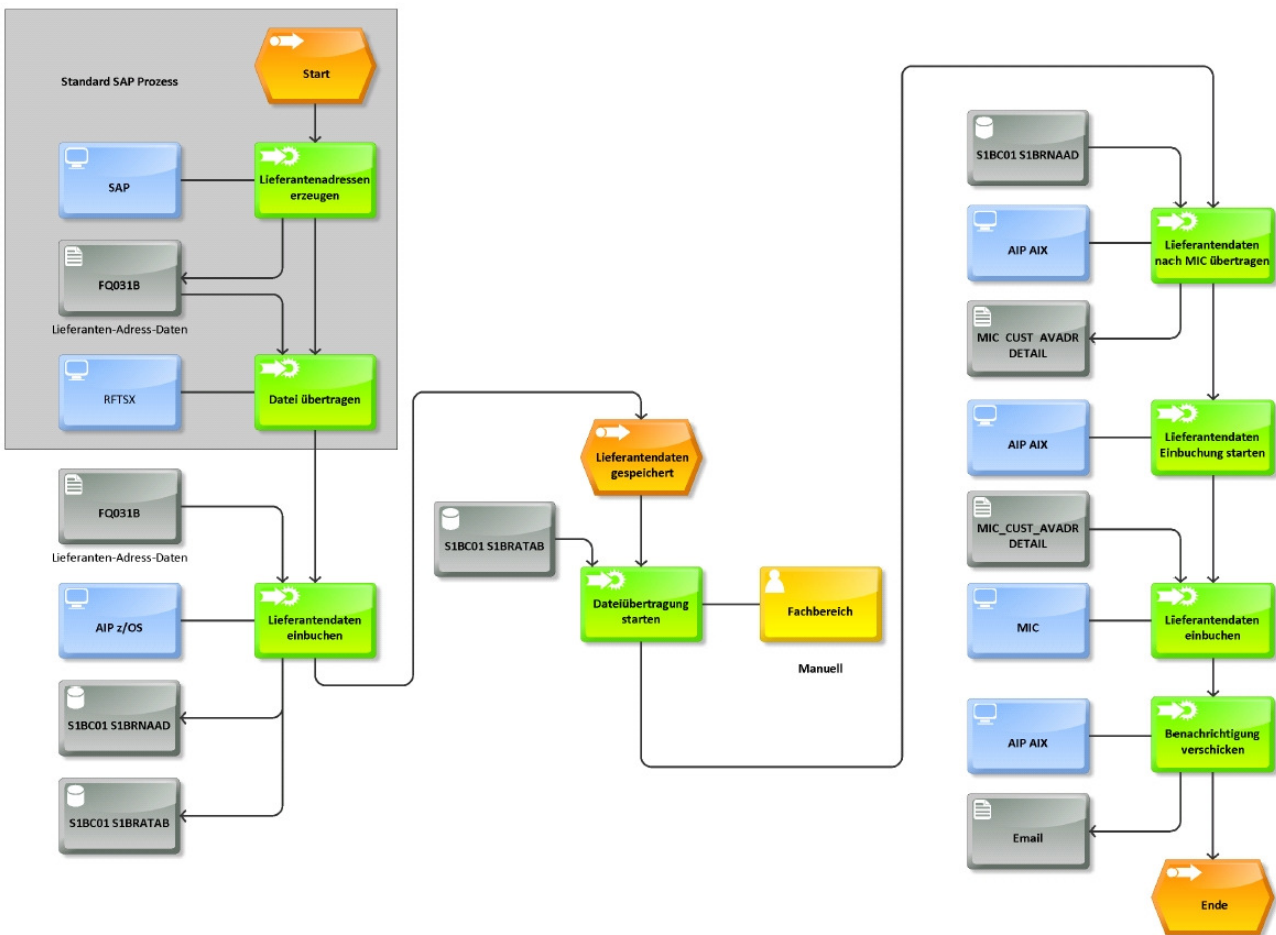


Abbildung 2: Prozessstruktur vor der Änderung

In diesem Prozess erzeugt ein Fremdsystem (SAP) eine Schnittstelle mit Lieferantendaten, welche auf die z/OS Maschine übertragen werden. Ein Batchprozess (Lieferantendaten einbuchen) verarbeitet diese Daten und schreibt sie in die Datenbank (DB2 z/OS). Eine Meldung wird in der Swing Applikation (Java Client) dem Benutzer

bekannt gegeben und er startet die Übertragung der Daten an die Zollsoftware, die außerhalb der Applikation die gleichen Daten mit einer anderen Aufbereitung benötigt. Ziel des POCs ist die Abschaffung des manuellen Teils. Das bedeutet, der Batchprozess („Lieferantendaten einbuchen“ in dem Bild) soll nicht nur die Daten aufbereiten, sondern sowohl die Meldung an den Benutzer schicken als auch die Datenübertragung anstoßen. Die Datenübertragung in dem Prozess ist aber ein JEE Baustein. Der einfachste Weg, um Java Komponenten zu verwenden, ist diese aus Java Programmen anzusprechen! Und damit haben wir eine Problemstellung gefunden, mit deren Lösung wir die Umgebung und die Technologie überprüfen können, also unser Proof-of-Concept. Das geänderte Prozessbild zeigt JZOS (ein Batch Toolkit für z/OS), an der Stelle des manuellen Schritts.

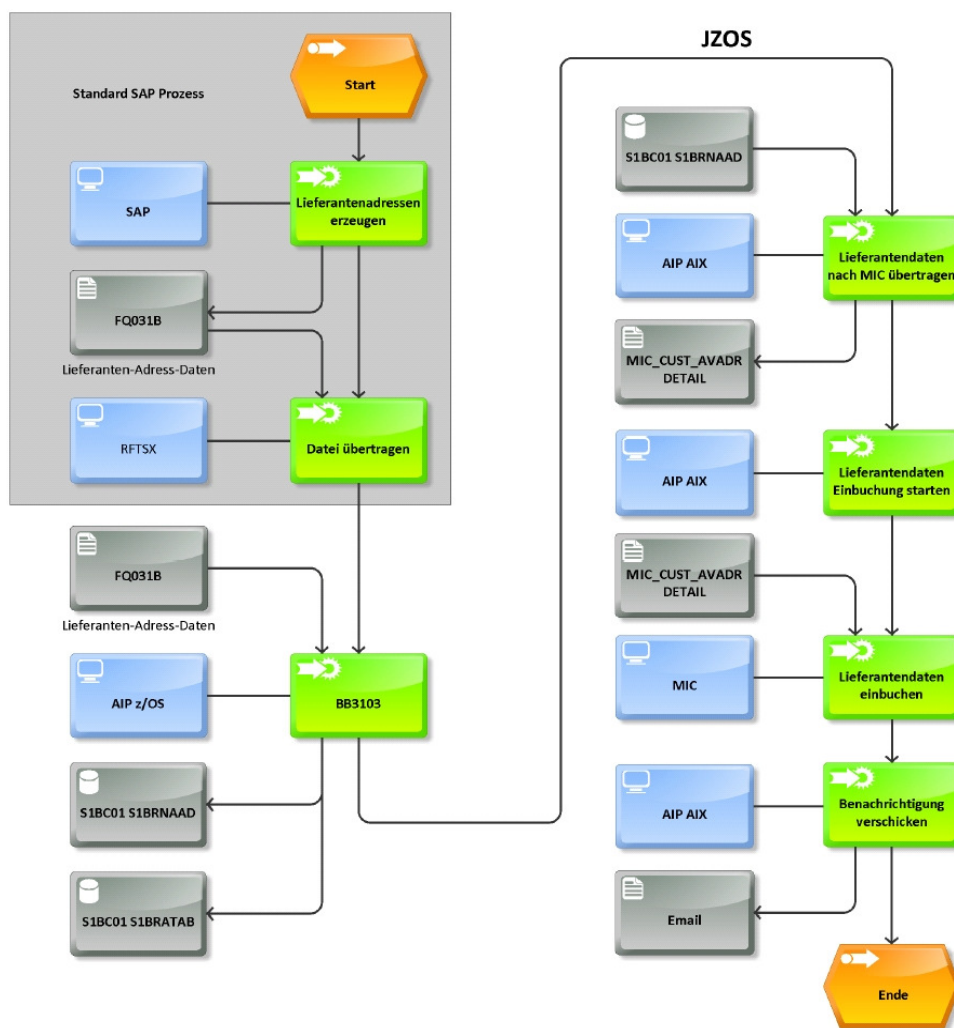


Abbildung 3: Prozessablauf nach der Änderung, ohne manuellen Schritt

Die Ablösung des manuellen Schritts wurde wie folgt implementiert:

- das vorhandene COBOL Programm im Batchprozess wurde erweitert, damit ein Parameter für das Java Programm erzeugt wird.

- der Batch Job wurde mit einem Java Step erweitert (JZOS).
- ein neues Java Programm wurde geschrieben, das die Java-Applikation auf AIX aufruft (dieses neue Programm läuft im JZOS Step).
- weil der JEE Teil in einer sicheren Umgebung läuft, wurde ein technischer Benutzer für den Ablauf angelegt.
- das neue Java Programm wurde für die Installation auf z/OS als tar-Datei ausgeliefert.

Das folgende Bild zeigt den Ausschnitt des neuen Steps im Job, wo das Java Programm aufgerufen wird.

```
//JAVA EXEC PROC=JVMPRC60,VERSION='60',
// LOGLVL='+E',
// JAVACLS='com.aformatik.java.host.lieferant.LieferantenUebertragungJaX
//          vaHost'
//STDENV DD *
. /etc/profile
APP_HOME=/usr/lpp/java/J6.0
export JZOS_HOME=/usr/lpp/java/J6.0/lib/ext

... Umgebung Konfigurieren ...

export CLASSPATH=".:$CLASSPATH":$MYPATH"
//MAINARGS DD DISP=SHR,DSN=BSA50.B.SL814A(+0)
```

Abbildung 4: JZOS Step im Batch Job

Eine vollständige Beschreibung des Jobs würde den Rahmen dieses Artikels sprengen. Deshalb hier nur eine kleine Einleitung in die Hauptpunkte:

- PROC:** für jede Java Version auf z/OS gibt es eine gelieferte Prozedur. In unserem Fall, haben wir die Version 6.0 verwendet (JVMPRC60)
- JAVACLS:** hier wird die Java Klasse angegeben, dessen „main-Methode“ aufgerufen wird.
- STDENV:** hiermit wird die „Shell-Umgebung“ konfiguriert (zum Beispiel, der Klassenpfad mit den notwendigen Bibliotheken).
- MAINARGS:** der Inhalt der Datei wird vom JZOS gelesen und als Parameter für die Java-Klasse aus JAVACLS weitergegeben. In unserem Beispiel hat das COBOL Programm diese Datei erzeugt mit den Informationen über die verarbeitete Schnittstelle.

Für die Entwicklung der Lösung wurde IBM Rational Application Developer eingesetzt. Eclipse wäre hier auch ausreichend, da man keine zusätzliche Funktionen des Rational Application Developer benötigt.

Die Installation des Java Programms auf z/OS wurde wie folgt durchgeführt:

- Sowohl auf einer Test LPAR als auch auf der Produktions LPAR wurde ein Verzeichnis eingerichtet, um die Programme zu installieren.
- auf dem Testsystem wurde für die z/OS Users (RACF Benutzer-ID) ein OMVS Segment eingerichtet und die entsprechenden Berechtigungen vergeben.
- auf dem Produktionssystem wurde ein Verzeichnis freigegeben und die Installation erfolgte mit Hilfe von Shell-Skripten.

Obwohl nicht zwingend erforderlich, ist ein Grundwissen über USS, FTP und JCL empfehlenswert.

Hier ein Teil des Java Code, der auf z/OS ausgeführt wird:

```
...
HttpClient client = new HttpClient();

List<String> authPrefs = new ArrayList<String>(1);
authPrefs.add(AuthPolicy.BASIC);

client.getParams().setParameter(AuthPolicy.AUTH_SCHEME_PRIORITY, authPrefs);

Credentials cred = new UsernamePasswordCredentials(benutzer, passwort);
client.getState().setCredentials(
    new AuthScope(host, Integer.parseInt(port), AuthScope.ANY_REALM), cred);

GetMethod get = new GetMethod(
    "http://<host>/<contextRoot>/<servlet>?<parameter_1>&<parameter_n>");
get.setDoAuthentication(true);

client.executeMethod(get);
...
```

Abbildung 5: Java Code auf z/OS

Die Parameter die JZOS aus MAINARGS gelesen hat werden einem Servlet übergeben und damit wird die Dateiübertragung gestartet.

Fazit

Mit dieser kleinen Prozessänderung haben wir

1. den Batchprozessor (JZOS) getestet,
2. das Deployment von Java Komponenten auf z/OS abgestimmt,
3. die notwendige Infrastruktur (OMVS/USS, FTP, usw.) überprüft und dokumentiert
4. und die Qualität des Ablaufs im Sinne des Kunden erheblich verbessert.

Einsparungspotential

Nach dem Proof-of-Concept wurde ein Test durchgeführt, bei welchem ein Prozess sowohl in COBOL als auch in Java implementiert wurde. Die Programme wurden mit folgender Logik implementiert:

- aus einer DB2 Tabelle werden 5000 Sätze sequentiell gelesen;
- für jeden gelesenen Satz wird über den Schlüssel einen Satz aus einer anderen DB2 Tabelle gelesen (Zugriff über Index);
- um I/O zu simulieren, wird der verwendete Schlüssel auf die Standardausgabe (SYSOUT) protokolliert;
- nach jeder Protokollierung wird eine Schleife durchgeführt, wo eine einfache Addition stattfindet, die auch protokolliert wird (um einfache Algorithmen mit I/O zu simulieren).

Erste Performance-Vergleiche zwischen beiden Lösungen haben gezeigt, dass die Laufzeiten kaum zu unterscheiden sind.

Anmerkung: Wir haben zwei unterschiedliche Implementierungen in Java ausprobiert. Mit dem JDBC Type 4 Treiber für die DB2 Verbindung war der CPU-Verbrauch (normale CP und zIIP zusammen) gleich wie bei COBOL, die Laufzeit aber viel länger. Mit dem JDBC Type 2 Treiber war der CPU-Verbrauch (normale CP und zIIP zusammen) wieder gleich, aber das Java Programm ist geringfügig schneller gewesen.

In dem Vergleich wird jedoch deutlich, welche enormen Einsparungspotentiale in der Umstellung stecken:

```
*****
* STEP: TESTC01 / 1 START: 09:48:38 AT 11 JAN 2011 END: 10:07.53 AT 11 JAN 2011 DURATION: 00:19:14 *
*-----*
* RESOURCE CONSUMPTION: CHARGES: * REGION= 999K * PAGING: * NOTES: *
* (EUR) * < 16MB: 10 216K * * *
* CPU SERVICE UNITS: 26 192352 * ALLOW: 9 704K * NVIO OUT: 0 * PGN : 000 *
* SRB SERVICE UNITS: 97143 * USED: 536K * NVIO IN : 0 * JOBCLASS: K *
* I/O SERVICE UNITS: 133399 * > 16MB: 1 498M * # SWAPS : 0 * COMPCODE: 0000 *
* MSO SERVICE UNITS: 0 * ALLOW: 400M * SWAP IN : 0 * # MOUNTS: 0 *
* TOTAL UNITS: 26 422894 * USED: 2 040K * SWAP OUT: 0 * CPU (SEC): 589 *
* CPU TIME : 000:09:46.05 145.16 * MEMLIMIT= 2G * VIO-IN : 0 * NORM-CPU: 542 165 *
* SRB TIME : 000:00:02.17 * ALLOW: NOLIMIT * VIO-OUT : 0 * IO-TOTAL: 133 412 *
* ZAAP TIME : 000:00:00.00 * *
* zIIP TIME : 000:00:00.00 * *
* TOTAL CHARGE >> 145.16 * USED: 0M * *
*-----*
```

Abbildung 6: Ressourcenverbrauch: COBOL

```

*****
* STEP: JAVAJVM / 1 START: 10:20:47 AT 11 JAN 2011 END: 10:35.34 AT 11 JAN 2011 DURATION: 00:14:47 *
*-----*
* RESOURCE CONSUMPTION: CHARGES: * REGION= 0M * PAGING: * NOTES: *
* (EUR) * < 16MB: 10 216K * * * *
* CPU SERVICE UNITS: 23 969634 * ALLOW: 16 384K * NVIO OUT: 0 * PGN : 000 *
* SRB SERVICE UNITS: 83884 * USED: 32K * NVIO IN : 0 * JOBCLASS: K *
* I/O SERVICE UNITS: 118614 * > 16MB: 1 498M * # SWAPS : 0 * COMPCODE: 0000 *
* MSO SERVICE UNITS: 0 * ALLOW: 784M * SWAP IN : 0 * # MOUNTS: 0 *
* TOTAL UNITS: 24 172132 * USED: 465M * SWAP OUT: 0 * CPU (SEC): 275 *
* CPU TIME : 000:04:32.74 67.56 * MEMLIMIT= NOLIMIT * VIO-IN : 0 * NORM-CPU: 253 322 *
* SRB TIME : 000:00:01.88 * ALLOW: NOLIMIT * VIO-OUT : 0 * IO-TOTAL: 118 628 *
* zAAP TIME : 000:00:00.00 * * * *
* zIIP TIME : 000:04:23.58 * * * *
* TOTAL CHARGE >> 67.56 * USED: 0M * * *
*-----*

```

Abbildung 7: Ressourcenverbrauch: Java

Obwohl der Prozess hier sehr einfach gehalten wurde, kann man gute Indikatoren aus den generierten Zahlen auslesen. Sehr interessant hier ist der Vergleich zwischen teuren CPs und zIIPs (je nach Kundensituation kostenlos). Hier nochmals die Zahlen:

	COBOL	Java
Laufzeit	19:14	14:47
CPU TIME	09:46:05	04:32:74
zIIP TIME	00:00:00	04:23:58
Kosten	145.15 EUR	67.56 EUR

Die Businesslogik, welche über die eingebaute Schleife mit Protokollierung simuliert wurde, läuft bei COBOL vollständig in den normalen CPs. Beim Java, in den zIIPs! Bei der Kostenberechnung (abgebildet im Protokoll) bedeutet das für diesen kleinen Batch-Ablauf eine Reduzierung um mehr als 50% der Kosten!

Ausblick

Als Ergebnis des POCs und der Performance-Messungen können folgende Ziele formuliert werden:

→ **Neue Batch-Prozesse sollen in Java realisiert werden.**

Begründung

1. die Entwicklungskosten sind niedriger, denn man vermeidet die vielen „Compile-Zyklen“ die bei der COBOL-Entwicklung anfallen und welche reine, teure CP Kosten sind.

2. Die Laufzeitkosten sind niedriger, da man „Special Engines“ (zIIPs und zAAPs) nutzen kann und die Java Programme auf diese Prozessoren ohne Veränderung laufen können.
3. Das vorhandene Know-how wird optimal ausgenutzt, da Entwicklungsteams immer mehr über eine solide Java Ausbildung verfügen.
4. Die neuen Programme werden mit IRAD oder Eclipse auf dem Client entwickelt (was wiederum zu CP Einsparungen im ISPF führt) und im gleichen Code Repository (SVN) abgelegt, was die Verwaltung vereinfacht.

→ **Vorhandene Java-Batchabläufe, die auf AIX implementiert sind, sollen bei neuen Releases auf z/OS ausgelagert werden.**

Begründung

- Das Zusammenlegen (auf Neudeutsch, „collocating“) des Java Programms und der Datenbank auf die gleiche Plattform bringt eine bessere Performance, weil damit der Netzwerk Overhead vermieden wird.

Zusammenfassung

Java auf z/OS hat sich als eine gute Alternative zu den COBOL, PL/I und Assembler Programmen erwiesen. Die Einrichtung der Umgebung ist sehr einfach und ohne zusätzliche Kosten zu konfigurieren. Und, last but not least, war das Endergebnis der Untersuchung mehr als zufriedenstellend: der CP Verbrauch ist vergleichbar, genauso die Ausführungszeiten. Jedoch kann mit der Konfiguration von zAAPs/zIIPs eine vergleichbar große finanzielle Einsparung erzielt werden. Voraussetzung dafür ist aber eine solide Ausbildung in den genannten Technologien! Wie gesagt, „Wissen ist Macht“ bzw. „Wissen wie's geht“, was speziell unser Motto ist. Bei weiterem Interesse sprechen Sie uns bitte an: www.aformatik.de

Warenzeichen

IBM, WebSphere, Rational, DB2, z/OS, AIX, RACF und System z sind Warenzeichen der International Business Machines Corporation in den USA und/oder anderen Ländern.

Java und Oracle sind Warenzeichen der Oracle Corporation und/oder ihrer Tochtergesellschaften.

UNIX ist ein Warenzeichen von The Open Group in den USA und/oder anderen Ländern.

Weitere verwendete Produktnamen, Warenzeichen und geschützte Warenzeichen sind im Besitz ihrer jeweiligen Eigentümer.