2011

JPMorgan Chase

ROBERT ZWINK , VP Implementation Services, Chief Development Office

[RUNNING OPEN SOURCE ETL ON A MAINFRAME]

Pentaho is an open source framework written in Java which includes a full featured Extract Transform Load (ETL) tool called Pentaho Data Integration (PDI). Programmers leverage PDI to create custom transformations which can be a direct 1:1 translation of existing COBOL. A rich palette of out of the box components allows the transformation to be assembled visually. Once finished, the transformation is a completely portable Java application, written in a visual programming language, which runs fully within a java virtual machine (JVM). Java programs created by PDI are 100% zAAP eligible.

Contents

ABSTRACT 3
GENERAL TERMS 3
INTRODUCTION 3
BACKGROUND 4
Assumptions and Requirements 4
Chargeback Model 5
A Push to Open Source 5
APPLICATION ARCHITECTURE 6
Mainframe Batch Processing 6
Data Access 6
File I/O 6
Versioning, Deployment, Scheduling and Monitoring 7
BPXBATCH, BPXBATSL, or JZOS? 7
Summary: Comparison of tools for launching batch Java jobs 7
IMPLEMENTATION 7
EXPERIENCES & LESSONS LEARNED 8
Billable CPU Reduction 8
Enhancements needed for direct MVS data access 9
Emerging Technology without Commercial Support 9
CONCLUSIONS 9
A Hybrid Approach 9
BIOGRAPHY 10
REFERENCES 10

ABSTRACT

Pentaho is an open source framework written in Java which includes a full featured Extract Transform Load (ETL) tool called Pentaho Data Integration (PDI). Programmers leverage PDI to create custom transformations which can be a direct 1:1 translation of existing COBOL. A rich palette of out of the box components allows the transformation to be assembled visually. Once finished, the transformation is a completely portable Java application, written in a visual programming language, which runs fully within a java virtual machine (JVM). Java programs created by PDI are 100% zAAP eligible.

GENERAL TERMS

COBOL, ETL, Java, Mainframe Modernization, zAAP, zIIP, z/OS

INTRODUCTION

In 2004 IBM introduced the IBM System z Application Assist Process (zAAP). The objective of the zAAP is to *enable integration of new Java based Web applications with core z/OS backend database environment for high performance, reliability, availability, security, and lower total cost of ownership* (Walsh). While the lower total cost of ownership is multi-faceted, a core component is an attractive pricing model designed to encourage customers to run Java workload at a reduced rate. The zAAP processor is dedicated exclusively to the execution of Java workloads under z/OS.

Java development on the mainframe is not a new technology. For years Java has executed on the mainframe under MVS, CICS, IMS, DB2 and WebSphere (Goetze). Success in these areas has prompted managers charged with reducing mainframe operating expenses to consider Java based alternatives to traditional development practices (Morris).

While reduced operating costs and a dedicated processor for mainframe Java applications is a compelling reason to re-engineer existing processes, in practice there are considerable ancillary processes which must change along with the programming language to be effective. In practice it is these ancillary processes which cause otherwise small cost savings initiatives to balloon past practical return on investment. Pentaho provides a low cost framework for developing java applications accessible to both mainframe system programmers and traditional Java developers.

Pentaho is an open source framework written in Java which includes a full featured Extract Transform Load (ETL) tool called Pentaho Data Integration (PDI). Programmers leverage PDI to create custom transformations which can be a direct 1:1 translation of existing COBOL. A rich palette of out of the box components allows the transformation to be assembled visually. Once

finished, the transformation is a completely portable Java application. PDI transformations are written in a visual programming language, which runs fully within a java virtual machine (JVM). Java programs created by PDI are 100% zAAP eligible.

Transformations are developed in a feature rich IDE, ideally on the developers own laptop. Moving development off of the mainframe onto the developers' own laptop also reduces mainframe CPU time during debugging. The resulting transformation is physically stored as an extensible markup language (XML) configuration file which can be easily transferred to a mainframe using standard file transfer protocol (FTP). Once on the mainframe the PDI runtime is invoked from job control language (JCL) with the XML file referenced for execution. Existing schedulers, version control and monitoring tools can remain in place. This configuration provides a fast and efficient way of re-engineering existing COBOL batch by leveraging a rich open source framework, while leaving in place ancillary processes which provide stability and complexity to the mainframe operating environment.

BACKGROUND

Assumptions and Requirements

A robust mainframe operating environment consists of a development, execution, and monitoring stack capable of executing high performance batch and online applications. While Java may be a viable replacement for COBOL, there is often a strong desire to leave other elements of the development, execution, and monitoring stack in place. Specifically in the field of mainframe batch processing this research assumed a strong desire to benefit from the Java programming language while continuing to leverage existing systems for scheduling, monitoring, code promotion, data access, security, messaging, defect tracking, transaction processing, and problem resolution. The primary strength of leveraging PDI on the mainframe is that these processes can be both left alone and therefore leveraged by the solution.

In order to execute PDI on the mainframe the operator must have IBM Java 6 and Unix System Services installed (USS). This research also assumes that the optional z:OS Toolkit installed in order to leverage the z:OS methods of executing Java from JCL. Any additional client jars must also be made available; this includes JDBC jars for connecting to IMS and DB2. This paper will cover executing PDI transformations using the IBM JZOS program originally developed by Dovetail Technologies which is part of the CO:Z Toolkit installed on most modern mainframe.

Finally, experience with JCL and JVM configuration parameters will be required to understand the technical examples illustrated in this paper.

Chargeback Model

An IT Chargeback Model provides monetization of IT resources. While there are many versions of chargeback, one based on consumption, as opposed to subscription or number of users, is ideal for this optimization technique. Consumption based chargeback models provides IT organizations a measurable way to charge more for application that use more, and charge less for applications that use less (Singh).

When a batch application is executed on the mainframe statistics are recorded about its runtime characteristics. The total amount of CPU time consumed by the batch application is typically recorded in SMF Type 110 records and later monetized in the chargeback process based on an hourly run rate. If a batch job required 1.5 hours of CPU time, and a CPU hour is charged back at $22 per hour, then the batch job would have cost $22 * 1.5 hours = $33 total. The approach described in this paper is intended to reduce the billable CPU time by offloading work from a billable variable cost CPU to a fixed cost reduced rate zAAP specialty CPU that is either not billed or billed at a reduced rate in a consumption based chargeback model.

A Push to Open Source

The decision to leverage open source technology on the mainframe must come with a deliberate evaluation of the impact to your organization. Total cost of ownership and the risk of community driven development is outside of the scope of this paper. While projects such as Apache Tomcat, Subversion and the Linux operating system have found strong footholds in enterprise software development, the decision to leverage open source is often part of a larger enterprise wide position. Open source ETL tools are an emerging alternative to traditional commercial vendors. If a strategic decision to leverage open source across the enterprise has been made, adoption of PDI on the mainframe is even more compelling. PDI offers a unique advantage of any other ETL tool, commercial or open source, because it executes as Java. At the time of this research there are no other alternatives to this unique capability.

This paper demonstrates the successful execution of PDI on the mainframe. It is believed by the author to be a novel approach, and at the time of this research, completely unsupported by the existing PDI open source development community. As Java and open source in general finds footholds on mainframe computers, this is expected to change. By leveraging this approach developers are assuming full responsibility for testing and debugging the framework. In light of the substantial cost savings made available by easily executing java on the mainframe, this may be justified. Regardless, it is important to note that a commercial version of PDI supported on the mainframe does not exist in any form. This research describes the necessary modifications to successfully execute PDI on the mainframe and the next steps needed in the community development to bolster its capabilities on the mainframe platform.

APPLICATION ARCHITECTURE

Mainframe Batch Processing

Mainframe batch processing leverages the Job Control Language (JCL) to execute programs. Traditional batch architecture is often layered, having data access controlled by a database management system or file system. Configuration information supplied to the batch job at runtime allows job portability across multiple environments.

Typical batch processing is initiated at regular intervals by an automated scheduler. The scheduler executes JCL which in turn controls the order or steps that a program takes to complete execution. Common steps found in JCL may extract data from a database, transform data by either sorting or applying business rules, and finally loading the data into a file, queue, or database.

When a mainframe batch step requires a robust programming language typically COBOL is chosen. COBOL is widely supported on the mainframe and accessible to mainframe programmers. Regardless many organizations have chosen to replace their existing COBOL development with the Java programming language. In both cases once a program is created within Java or COBOL, it is executed by a job step, data is accessed by the program, the data is manipulated, and then data is persisted to a database of file. While this does represent a broad generalization, the overall pattern is pervasive in mainframe batch processing.

Data Access

Data is typically stored in a file or database. Access to the data is provided in many ways, such as directly by the file system or via robust network protocols which provide client/server access to data via queues and services. Database management systems (DBMS) also provide robust application programming interfaces (API) for local and remote database access. While there are many ways to access data on a mainframe, it is particularly important that a programming language provide this support. PDI is written in Java with a robust mechanism for leveraging existing client libraries.

File I/O

While Apache VFS is integrated as the primary data access API, the ability to extend PDI using custom Java classes provides for easy access to VSAM and MVS datasets using IBM Java Record IO (IBM: Java Record I/O overview.).

The Apache VFS API is an abstraction to the filesystem which provides for uniform file naming to disparate operating systems. Apache VFS does not directly support MVS datasets existing outside of the HFS Filesystem on mainframes. For PDI to natively handle all mainframe file access on MVS and extension to VFS would be required.

Versioning, Deployment, Scheduling and Monitoring

Change management systems vary from site to site. Changement, CVS, and Subversion are but a few of the source management systems used by mainframe computers. PDI generates XML configuration which contain complete program logic. The only requirement of PDI for change management systems is that it support XML files.

BPXBATCH, BPXBATSL, or JZOS?

There are three ways to execute java as mainframe batch. IBM provides a succinct overview for comparison purposes (Goetze). A chart from that article is reproduced below.

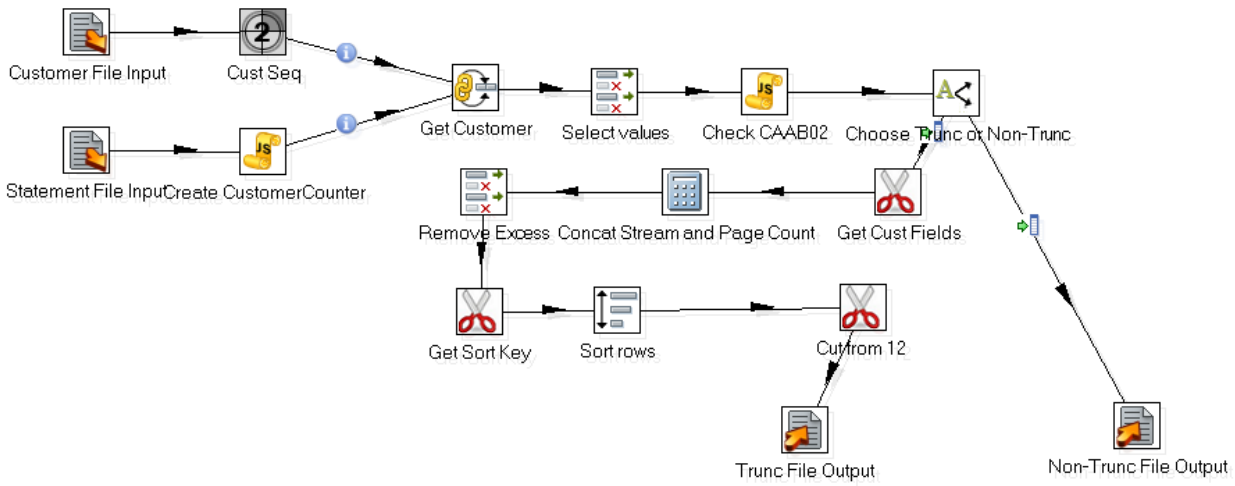
	BPXBATCH	BPXBATSL	Custom JVM launcher (JZOS)
Flexible configuration of environment variables	Yes, variable substitution and scripting are not allowed	Yes, variable substitution and scripting are not allowed	Yes
Route output directory to SYSOUT datasets	No	No	Yes
Control output encoding separately from default JVM encoding	Yes, using iconv	Yes, using iconv	Yes
Condition-code passing between Java and non-Java steps	No	No	Yes
Use MVS datasets and DD statements	No	Yes	Yes
JVM runs in same address space	No	Yes	Yes
Communication with MVS console	No	Yes, using _console function from a JNI routine	Yes

Summary: Comparison of tools for launching batch Java jobs

This chart demonstrates the flexibility of the JZOS launcher provided as part of the CO:Z Toolkit. JZOS simplifies the JCL, runs in the same address space, and cause all the java processing to run on the zAAP specialty engine. Practically speaking JZOS starts the JVM without the Unix System Services address space. The other two approaches will cause the USS to execute albeit briefly with no discernible value.

IMPLEMENTATION

A mainframe programmer using PDI develops applications using a visual programming language. While this is contrary to procedural COBOL programming, the effect is the same. In practice the visual programming language is self documenting. An example program is illustrated below.



This program is a direct port of a cobol application which reads two fixed file inputs and then pairs the two files similar to folds together two decks or cards. Business logic is evaluated, and based on the evaluation data can take two different paths to separate output files. One output file is essentially the combined data verbatim, while the second output file contains slight modification to the original data.

This transformation was created by reading the original COBOL code, following the programming language and leveraging existing component of the PDI framework. This research does not cover all of the nuances of the PDI visual programming language, through the author recommends the book written by the principal developer of PDI Matt Casters “Pentaho kettle solutions building open source ETL solutions with Pentaho data integration” (Casters).

Once complete the visual program is stored in an xml file containing all necessary configuration information. The PDI xml file can then be transferred to a mainframe HFS file system using FTP in binary mode which maintains the ASCII character set. The author found it practical to test the PDI xml file from Unix System Services prior to scheduling the Job to execute using JCL. Once satisfied the PDI program can be directly executed using the JZOS java launcher.

EXPERIENCES & LESSONS LEARNED

Billable CPU Reduction

Direct billable mainframe batch CPU reduced by 75% based on tests executed which compared COBOL and comparable PDI transformations. This is due solely to the zAAP offload eligibility of the Java programming language and DRDA DB2 zIIP offload eligibility created by the JDBC connection to DB2. The author believes this workload represents typical mainframe batch based on experience in financial data processing. Removing the DB2 requirement of the test

caused complete zAAP offload thus 98% reduction in direct billable mainframe batch CPU. Since direct billable mainframe batch CPU is the single largest component of mainframe batch processing a significant cost savings opportunity is made available.

Enhancements needed for direct MVS data access

PDI does not ship with native MVS file IO capability. The PDI XML configuration files needed to remain on the HFS filesystem, as well as any output files. Managing the transfer of HFS files to MVS is not ideal, thus the ability to write directly to MVS is a needed enhancement. Since PDI is fully open sources, as well as the Apache VFS libraries for output, it is possible to add this functionality. Until this happens, juggling between HFS and MVS is a necessary workaround.

Emerging Technology without Commercial Support

Mainframe batch processing using PDI is new and completely unsupported. The PDI community forums have no mention of this, and Pentaho commercial sales and support considers the mainframe completed unsupported. This means that shops willing to implement PDI as a potential cost save need to consider support in the total cost of ownership equation.

CONCLUSIONS

A Hybrid Approach

The experiences of this research all points to a hybrid approach for future PDI development on the mainframe. Based on this research critical COBOL processing with high levels of DB2 access should remain in place primarily due to lack of commercial support. Non-critical non-DB2 COBOL programs however are great candidates. Environment build jobs, backup jobs, and other maintenance tasks especially in development mainframe environments should be considered for this unique approach. As development continues with PDI, open source contributions by large mainframe shops will allow for a shared library of mainframe specific components, as these develop the case for PDI will also. If a commercial support offering for PDI on the mainframe emerges, the case may also be stronger.

BIOGRAPHY

Robert Zwink works on a dedicated team focused on ensuring continued technology and process innovation across large and disparate systems at one of the world's largest financial services firms. Prior to this role he spent two years in as manager of mainframe application performance and capacity planning. Today, he creatively applies his experience designing and implementing complex enterprise server based solutions to mainframe modernization projects.

REFERENCES

alphaWorks : IBM JZOS Batch Toolkit for z/OS SDKs : Overview. (n.d.). alphaWorks : Emerging technologies. Retrieved August 5, 2011, from http://www.alphaworks.ibm.com/tech/zosjavabatchtk/download

Casters, M. R., Bouman, R., & Dongen, J. v. (2010). Pentaho kettle solutions building open source ETL solutions with Pentaho data integration. Indianapolis, Ind.: Wiley.

Fowler, M. (2003). Patterns of enterprise application architecture. Addison-Wesley Professional.

Goetze, Steve, and Kirk Wolf. "Java batch jobs on z/OS and OS/390." IBM - United States. N.p., 5 Mar. 2005. Web. 25 Apr. 2011. <http://www.ibm.com/developerworks/systems/library/es-java-batchz.html>.

IBM: Java Record I/O (JRIO) overview. (n.d.). IBM - United States. Retrieved August 5, 2011, from http://www-03.ibm.com/systems/z/os/zos/tools/java/products/jrio/overview.html

Morris, Robert. "Specialty Engines: How to Reduce Costs and Save MIPS zIIP/zAAP Specialty Engine: Specialty Engines: How to Reduce Costs and Save MIPS ." MainframeZone. N.p., 29 Apr. 2010. Web. 25 Apr. 2011. <http://www.mainframezone.com/applications-and-databases/ziip-zaap/specialty-engines-how-to-reduce-costs-and-save-mips/P2>.

Singh, Lisa. "Consumption-based chargeback model: For Northrop Grumman CIO Bernie McVey, a more efficient IT organization starts here." ExecutiveBiz. N.p., 23 Aug. 2010. Web. 26 Apr. 2011. <http://blog.executivebiz.com/2010/08/consumption-based-chargeback-model-for-northrop-grumman-cio-bernie-mcvey-a-more-efficient-it-organization-starts-here/>.

Walsh, Kathy. "zIIPs and zAAPs: Everything New and Old." SHARE. IBM. Austin Convention Center, Austin. 1 Mar. 2009. Lecture.