Cobol to Java Conversion
THE BUSINESS CASE FOR COBOL TO JAVA CONVERSIONS

Much has been said lately about the age of the Cobol programmer and the coming worldwide staffing crisis. This paper looks at how this problem will impact businesses that run Cobol based applications and offers a solution that will eliminate the problem entirely and open legacy applications up to modernisation.

No major organization is currently offering a viable, low-risk solution to the problem. They will perform a manual rewrite, often supported by use of business rule mining tools. While this, in theory, can produce a better eventual result than automated migration, the risk of applying a significant manual effort to a ‘moving target’ problem is unacceptable in most cases.

MSS has created an automated migration path to move mainframe Cobol applications to JEE Java that can be run on any compliant platform including low cost industry-standard Unix and Windows and very low cost open source commodity hardware.

THE PROBLEM

There are about 200 billion lines of Cobol code in the world and it is said that this is increasing at a rate of 5bn lines per year. Many of these applications have their origins in the 1960 when Cobol was first used in mainframe applications. There is a lively debate going on about the future of Cobol.

A recent article sited in the Financial Times stated that half of mainframe and Cobol programmers are over 50 years old, i.e. the average programmer is 57. This means that in five to ten years there is likely to be an increasing shortage in Cobol programmers that could impact business in a number of ways:

- Increase in the cost brought about by skills shortage
- Longer time to market for new products that require Cobol skills
- Difficulty in maintaining Cobol based applications especially where changes in legislation and compliance demand changes in business processes
- The low number of young people willing to be trained in Cobol programming will make this a growing problem for companies that use this language.

This will initially impact on the lesser-known Cobol dialects such as Unisys and VAX who already face a skills shortage because of the relative obscurity of the operating environment. Larger corporations may find that they can push the problem out but there are many smaller and medium sized companies that use mainframe computer who will experience the problem first, if not already. Several mainframe users we have spoken to have already sited this is a strategic issue for their businesses.
MSS’s migrate!COBOL tools offers JAVA as a destination language. Because the tool is based on mature compiler technology - MSS has been performing Cobol migrations for 30 years - the product is of a sophisticated, modular design and can handle multiple dialects of Cobol.

Initial projects have targeted Unisys Cobol and have enabled (Libra) mainframe to open systems migrations. See appendix A for success stories.

The generated Java code is as maintainable as the Cobol code was and in fact is mostly a line for line conversion of the original Cobol application. The end user of the application will notice very little difference when working on the application and will require very little training, if any at all when using the modernised interface. Most if not all mainframe sites run Cobol applications.

MSS’s Cobol to Java migration tool provides these clients with four essential benefits:

- True modernisation of the legacy application including replacement of terminal emulation with native web browser access
- Access to a young and enthusiastic work force that insist on developing in modern language
- A massive increase in development productivity allowing the business to react faster to market trends
- A massive reduction in the running costs of their core business applications.

Carrying forward the vast investments made in the legacy application

The technical challenge is to create a solution that is efficient, compact and maintainable. Of the tools on the market that claim to be able to convert Cobol to Java the best one imposes a heavy penalty on the programmer - up to 100 source files per program and 10x the original number of lines of code need to be maintained.

The MSS Java migration tool, by contrast, does not increase the overall line count by deploying innovative approaches to migration problems. Some of these are;

DATA DECLARATIONS

Cobol record structures are, effectively, groups of bytes that can be viewed in different ways. A date can be viewed as a whole or as a group comprising year, month and day. Just the month element isolated and manipulated - and it will affect the value of the containing group with the effect rippling up through all levels in the record structure.

The MSS solution is able to simulate exactly how records are defined in Cobol with data grouped together at multiple subordinate levels, implicitly remapping actual memory areas by defining a group object that can be built up in the runtime initialisation phase from
‘elementary’ items with Cobol-like attributes and/or other groups. The methods of the groups provide all the record handling functions available in Cobol.

**PROGRAM STRUCTURE AND LOGIC FLOW**

Procedure division code is divided into SECTIONS and PARAGRAPHS which can be accessed by PERFORM and GO TO statements. This presents a considerable problem if approached from a conventional point of view since Java has no GO TO construct.

The MSS solution is to view each paragraph as an object in its own right which can be passed to a method that can control its execution (strictly speaking, they are methods within a section; the section class uses reflection to “objectify” these methods so that they become independently addressable so that gotos and performs can be applied to them). Thus, the perform or goto methods can be passed paragraph names to execute. The sequence of execution is determined from analysis of the program structure by the program itself using Java’s reflection features.

The net result of this is that the Java program structure is exactly analogous to the original Cobol program structure and immediately recognisable by the original Cobol programmers. This allows existing Cobol programmers to be easily retrained in Java thus preserving the most valuable asset of any IT department, the application knowledge of the programmers themselves.

**SOFTWARE ENVIRONMENT**

The Java programs created from batch programs are POJOs (plain old java objects). Each will run in its own JVM and will generally be controlled from perl (or ksh) scripts created from the job control programs of the original environment.

Online programs will be created as EJBs (Enterprise Java Beans) with MSS middleware supplied as necessary:

- MCS!Lite is a TP Monitor that can route messages, based on message content or explicit routing, between workstations and programs or programs and programs.
- WebManager is a session controller that provides a graphical user interface from a browser into the system and enables screen formats defined in XML to be interpreted at runtime and processed using CSS and XSLT to effect a desired look and feel.
- The JEE infrastructure in which the programs are run can be based on any conforming middleware - WebSphere, JBoss, WebLogic etc.. In all cases the JEE platform will allow distribution of the application over several physical servers as necessary to provide the required performance. And, of course, the underlying hardware can be anything that runs a JVM, typically Linux, Unix or Windows but also zLinux or z/OS.
- The database can be any relational database. MSS has software to create a SQL schema from a mainframe database as well as extract and load processes to populate the new database. Cursor routines are generated if necessary to simulate mainframe sequential record processing.

The following diagram represents the new software environment:
XML COMMUNICATION

XML files are generated to define the data layouts of the screens (one file per screen). The XML files are processed under the control of a single XSL file to create standard HTML which is sent to the browser. Included in the HTML are JavaScript routines to perform basic validation and style information utilising a CSS file. The CSS file can be modified by any standard tool to change the look of the screens globally so that no individual changes to forms are necessary. Similarly the XSL file can be changed to change the “shape” of the screen (e.g. mapping of column/row to pixels, adding GUI widgets).
SCREEN FEATURES

The screen below is a fully automatically generated screen from a mainframe form, with the exception of an additional “email” field allowing lowercase characters.

This illustrates the following features:

1. Optional “field information” bar which indicates type of field as you move through the form.
2. Optional clickable “mouse button” bar in addition to transmit keys, etc.
3. Automatically filled in decimals (PRICE and DISCOUNT) as you type.
4. Automatic case conversion during input. Fields may be configured to accept lowercase characters (e.g. the eMail field above).
5. Automatically filled "as you type" leading zero's in configured fields (e.g. TOTAL).
6. Automatic "implied decimal" as you type (e.g. TOTAL)
7. Optional negative numbers in red feature.
8. Pop up warnings and error messages occur as you type. The messages are unobtrusive in that they don't require clicks or additional key depressions. They appear and disappear as you move through the fields. Fields in error will prevent transmission of a form.
9. Fields in error are highlighted in red and fields with warnings in a blue colour. This particular warning field indicates that after automatically inserting the required decimals the value was truncated.
10. Optional clock (which can be positioned in any corner).
11. Current field is highlighted (border and background colour change).

In addition to the features visible in the screenshot there are a variety of others:

12. Optional automatic screen re-sizing. The browser window can be set so that it's always the correct size, with no scrollbars.
13. Optional "move to next field" when typing fills current field.
14. Optional debug information window for developers.
15. Optional real-time statistics panel (shows transaction time and other relevant information).
16. Automatic calculation for arrow key navigation. i.e. up and down arrow keys will move the cursor to the field you intuitively believe it should.
18. Fully configurable transmit, reset, help keys.
19. Fully configurable function keys. i.e. If your application uses function keys to input certain data into a particular field and then perform some action, this can be done. Note that only the Firefox and Microsoft Internet Explorer browsers fully supports this feature.
20. Input error checking as you type. If a field is unsigned, it won't allow input of negative numbers, etc. This feature only works in Firefox and Internet explorer. Other browsers do not allow the prevention of input characters.
21. Entire form definition is in XML.
22. Multiple scheme support for CSS file defined schemes.
23. Optional user configurable colour schemes.
24. System level configurable defaults include options which allow or disallow users from configuring their own options. Options may be individually allowed/disallowed.

If allowed, the additional information panels (button bar, debug window, field information bar and statistics bar) may be toggled on and off at will. The system administrator can prevent or force users from either toggling or viewing any of the bars.

---

THE ADMIN CONSOLE

The Java migration path uses a full-featured front-end system that is very flexible and has many variations. To assist in configuring these options, MSS provide an administration console as part of the product. The console uses AJAX techniques in order to present required information on demand. For example, if the Security manager->LDAP manager option is selected, additional fields will appear which are related to the LDAP option.

The admin console deals with System Parameters and Application Parameters as well as providing a convenient place to handle function key mapping.

To access the console, simply press F5 while logged on as an administrator.
The System Parameters Screen appears thus:

The global properties loader specifies how often the application should refresh the properties. There will be some delay - otherwise the overhead could be high - but changes made in the admin console will be reflected in the operation of the application in a short time without having to restart.

The security layer is, perhaps, the most important feature of the console. Mainframe users are used to having few tools to integrate with enterprise security and typically rely on a home-grown password protection system embedded in the application itself. In the meantime, the enterprise itself has standardized on LDAP or Active Directory for general user access. The security layer controls allow an administrator to specify which (of several common choices) flavour of security to use and, at a detail level, enter the required parameters. This, for most installations is a major step forward in securing a vital resource.

The machine name binding section allows an administrator to describe how he wants the names of stations to appear to the application. Often the station name is used within the application for purposes such as an extra check on access to sensitive data, i.e. terminals in insecure locations should be denied access. The machine name binding allows this to be carried forward in the new environment.
The personalization section simply allows an administrator to let individual users personalize their own machine.

The **Application Parameters** screen is presented thus:

![Application Parameters Screen]

The Company Parameters section allows the administrator to document the application’s business environment and customize it for different companies (or divisions within a company). In the event of an application error, an error page is displayed along with the Support message and other useful information.

The Server Parameters section configures the application in terms of its look and feel (customizable CSS/XSL/JSP), the data source, and the way it functions for a user (the logon screen and the logoff action).

The **Key Bindings** screen is presented thus:
This screen allows the administrator to customize the key bindings. Key binding profiles may be created and selected within the xml for a particular screen. This allows different sets of function keys to be mapped for different user groups or for different logical groupings of forms. The labels (shown under the function key name) will be shown underneath the application form providing the “Show function keys” attribute is checked in the Application Parameters screen.

ERROR HANDLING

In addition to the log files, system errors are presented to the users using a form which shows relevant information can be used to directly email support. This includes whatever instructions were specified in the Application parameters admin screen. The user is presented with options to retry the form, restart the application, log off, send an email to support or view the full stack trace.

This sample error form was produced by shutting down the database during a commit operation.
WebManager conforms to the standard Model View Controller (MVC) architecture. The WebManager J2EE front-end can run in the same J2EE container as the EJB's or can be configured to run on a separate server. Communication between the two is via RPC for performance, passing a single parameter, again, defined by the XML form.

The EJBs correspond to the original transaction code and typically contain methods for initial screen display, input validation and input processing code. The EJBs themselves exist as stateless session beans which are managed by the container. Several threads corresponding to one transaction may exist at any one time if the traffic demands it. The
threads are available for multiple transactions and are only discarded after a period of inactivity in order to reduce the overhead of start-up, primarily establishing database connections.

The WebManager is capable of driving alternate back-ends simultaneously. For example, a transaction could be developed or migrated to Oracle PL/SQL as well as Java. To use the PL/SQL backend for a particular transaction it is simply a matter of naming the PL/SQL package in the XML form. Adding other back-end types is merely a matter of providing a single Java class for managing the backend.

Because all communication is defined in terms of XML the process achieves a high degree of inter-operability. WebManager is able to call (say) a login screen and run a Java EJB which does a recall to a PL/SQL online module which then, in turn, causes the next online module to be run as an EJB, which in turn calls up a Cobol screen and sends the message via MCS! Lite. The XML forms definitions for the PL/SQL implementation and for the Java EJBs are the same - a simple declaration in the XML form itself would cause a call to be made on the EJB, PL/SQL or Cobol module as required.

Added to this, calls made to other web services can be integrated as a project-specific customisation.

COMPATIBILITY

All development has been done using the SunW J2EE-RI Reference container, the reference implementation for all J2EE technology. This means that all 3rd party J2EE products (such as IBM WebSphere, Oracle Application Server, JBoss, Tomcat, etc) have to conform to at least this standard. The migrated code will therefore run out of the box on any J2EE platform. The only restriction is that you use Java 1.5 or later. This is because we have utilised the new “Generics” feature of Java. This shouldn't pose any difficulty since every major vendor has already moved to this version. If authenticating against Microsoft Active Directory, the minimum Java version must be 1.5.08.

No commercial or open-source frameworks are used in the code. This means that the solution is vendor-independent and insulated from future changes in this volatile market.

DEPLOYMENT

We supply ant scripts for compiling the application and creating an EAR file which is compatible with JBoss. All database connections are via container managed data-sources. The resultant EAR file can be directly deployed to the application server. Small modifications to the ant scripts may be required to deploy to other application servers, such as WebSphere.

ONWARD DEVELOPMENT/REFACTORING OF THE APPLICATION
The application is left in a form that is very standard so that full advantage of the latest development tools can be taken. The development tools are outstanding in their functionality and ease of use. It is possible (and easy) to single-step and set breakpoints throughout an entire transaction - even moving seamlessly from the web front-end to the online program, viewing and altering data as you go.

Each form is defined in a single XML file. This form is processed by an XSLT file and displayed on a standard browser with a CSS style sheet. XSLT is a processor (standardised by W3C) which uses the XSL definition file to transform XML to anything else - in this case, to HTML. What this means is that multiple XSL files could be prepared, each designed to produce transformations to different media, such as wireless or print, and the same programs will then be available without any re-coding. The CSS file can be modified to alter fonts, colours and backgrounds while the XSL file can be modified to introduce new features (such as drop down boxes). Different systems (e.g. DEV, TEST and LIVE) can be given different colour schemes etc. The possibilities are endless.

Integrated development tools can be used to good effect to manage future development. Oracle JDeveloper and Eclipse are used in-house and provide many useful features for assisting with code refactoring and development.

The overall design of the migrated code lends itself to incremental development. A single library routine can be altered, or changed to use a better Java style code without affecting other code. New EJ Bs can be developed and tested, then when complete, can be incorporated into the main system by merely changing the “bean” name in the XML descriptor file.

**RELIABILITY/SCALABILITY**

Another huge benefit of the technology is reliability and scalability.

An online program can crash without affecting any other users and an individual (failing) machine can be removed from the network without shutting down the system. If configured correctly users would not even notice a server crash.

The options for scaling up are no longer limited to increasing the size of the primary server. The container managed applications can be moved to different hardware, possibly running different operating systems, without any change or re-compilation. It is possible to have the online programs simultaneously running on large Unix servers, Linux PC and Windows boxes, all talking to a database which could reside on any number of machines (also with different operating systems) and multiple web-servers. Increasing capacity is merely a matter of adding a new machine to the network, configuring the J2EE container and loading your ear file. It is not even necessary to bring the system down to accomplish this.
APPENDIX A - SUCCESS STORIES

The case studies below describe projects where the automated migration of Cobol to Java was a key component, if not the key component in the conversion exercise.

The projects went live with no issues due to the automated nature of the tools involved.

KIR UNISYS TO LINUX MIGRATION CASE STUDY

BUSINESS PROBLEM

Krajowa Izba Rozliczeniowa S.A., or KIR as it is known, is the national clearings organization of Poland, one of the largest and most mature economies of the former Eastern Bloc.

KIR's main function is to act as a central point for receiving transactions (representing cheques) from banks and passing them to the appropriate destination banks. Because of the high usage of this method of payment in Poland the workload and necessary response time is extremely demanding.

KIR used a Unisys Clearpath Libra 690 mainframe running a Linc and Cobol application with significant Algol and WFL programming. This was relatively expensive and inflexible.

KIR wanted to move to a more modern platform that would be in line with current practice and provide more flexibility and better staff availability - abundant good programming resources are to be found in Poland but not in legacy languages. In addition, there was pressure to reduce cost of operation.

BUSINESS SOLUTION

KIR approached MSS requesting a proposal for a migration from the Unisys mainframe and LINC/Cobol/DMSII to Java/Oracle on a Linux open source platform.

The cut-over went smoothly and all of the performance goals were achieved. All of the original interfaces were reproduced exactly so that none of the large variety of users was affected.

In evaluating the project, Tomasz Jonczyk, the IT director said "Overall I feel that the migration project went extremely well. The fact that your tools automate the code conversion in every area - not just the Cobol and Linc but also the WFL and Algol - it went much faster than it would have done otherwise. And, I think, produced a better, more consistent result for us than could have been achieved in any other way."

OVERVIEW

KIR wanted to replace their Unisys Libra mainframe with one that would be more modern and flexible, perform better and be more cost-effective with no loss of reliability. Because of the stringent requirements of the National Bank of Poland and of the user banks, this was a key requirement.

The chosen target environment was Linux running on VMware virtualized HP servers. The virtualization software allows for single system images to be presented even though the physical hardware was located in a primary and disaster recovery site several kilometers apart. Failover, in case of any system failure, is handled automatically.

MSS used the migrate! series of products to transform the Linc system into a JEE Java application and to convert the Cobol file handling programs and the Algol technical libraries to Java. A significant body of WFL job control code was also migrated to perl scripting to automate the daily operation.
The DMSII database was ported to Oracle and code provided to load and access the Oracle data in the same way as the original. The performance of the database was optimized to give at least equivalent and in most cases better response than DMSII.

The migrate!LINC toolset was used to automatically migrate the Linc screen presentation to JSP supported by MSS’s WebManager software with business logic implemented as Java EJBs. For the batch system, freestanding Java programs, each running in its own JVM, were generated.

The Cobol programs at the heart of the operation were used to process files of transactions and sort and send on transaction records. migrate!COBOL was used to convert these programs to Java for compatibility with the migrated Linc. The resulting code was structured so as to be easily understood and maintained by the original programmers with minimal retraining in java. The developers received training in the Eclipse Java IDE the use of which made them very productive in a short time.

The files received from users are of a quite complicated format being a mixture of ebcdic, ascii and binary data with a digital signature calculated by Algol routines. These files have to be handled in a very efficient way since the time taken to process them is critical. As well as processing the files, financial statistics must be gathered from them and reports made to the National Bank of Poland to strict time deadlines. All the Algol routines were converted to Java with no loss of efficiency.

The scale of the migration was moderate 350 Linc Ispecs and 510 Linc Reports, 30,000 lines of Cobol code and 26,000 lines of Algol code. The complexity, however, was rather high centring on the complex file formats for which special Java routines had to be built.

KIR makes use of many features of ROC to handle print files and these were all replaced with appropriate scripting and browser-based file viewing and control.

CHALLENGES

The Cobol programs handled very complex file formats using unusual mainframe file types and a mixture of ebcdic, ascii and binary data in various layouts. In addition the digital signature calculation required constructing an ebcdic image of each record even though the processing was done on ascii data. An extensive library of Java routines was built to be able to cater for virtually any combination of character set and binary data.

KIR use ROC extensively for controlling and viewing print files. The standard Linc to Java implementation allows for flexible naming and placement of files but KIR’s requirements went beyond this. Configuration files were created to set defaults for file expiration and archive - current files kept in one directory and then moved to another after a certain time - and scripts created to action these. A browser-based print file viewer was implemented (based on the open-source sfm tool) and this is also used to control file expiration and archiving.

To aid system operation an ODT-like control system was developed so that the operations staff could quickly be notified of any issues with file processing and take action. The ODT facility handles MCP style messages and waiting entries.

RESULTS

The migration project was completed in 13 months and went live in June 2009. The system ran well from the start and showed improved performance.

KIR is extremely pleased to have the new system implemented in much cheaper commodity hardware but with the improved flexibility and resilience offered by the virtualized solution.

Not only is the application running better and at less cost but the developers are more productive using Java and Eclipse and, after training and initial experience, completely comfortable with Java as a language.
BUSINESS PROBLEM

Toronto Police Service is one of the largest municipal police forces in North America, serving the City of Toronto. Toronto, the largest city in Canada, has a population of more than 2.5 million residents, and serves as Canada’s economic capital.

The CASC system is a critical legacy system that ran on a Unisys mainframe. CASC is used by Toronto Courts, City of Toronto and TPS to schedule and manage police officers’ court attendance. First implemented in 1986 using Unisys LINC and COBOL, CASC was a prime candidate for a technology refresh in 2007.

TPS faced the usual problems that arise when new technology becomes the mainstream in an organization, leaving the mainframe applications as an expensive and isolated legacy. The problems were a decrease in available technical expertise and high operating cost and a need for increased flexibility.

Citing these issues, TPS requested proposals to migrate CASC to a modern platform.

BUSINESS SOLUTION

Toronto Police issued an RFP for a migration from the Unisys mainframe and LINC/Cobol/ DMSII to Java/DB2 on an IBM Unix platform.

The migration project was accomplished and went live in May 2008. The implementation was deemed to be very smooth, with minimal impact and with the online user-base (up to 6000 with 80 concurrent users on average) barely affected.

It was noticed straight away that performance in certain areas, especially the batch work, was substantially improved while the costs had diminished. Toronto Police are now experiencing significant savings in support costs and direct on-going IT costs. They have a more productive system than before and are well-positioned for future development.

OVERVIEW

In 2007 Toronto Police decided to lower their cost base by moving to Unix equipment with MSS International’s migrate!LINC and migrate!COBOL tools. MSS responded to their RFP addressing the issues.

MSS won the bid with the lowest cost and lowest risk approach - an offer of a fully automated migration from LINC and COBOL to Java (J2EE), XML and DB2.

MSS performed the core migration with Toronto Police testing the resulting code to their satisfaction.

The DMSII databases were ported to DB2 and standard database development tools already in use at TPS were used. The migration project took 9 months and involved a team of Toronto Police staff of up to 4.

The migrate!LINC toolset was used to automatically migrate the Linc screen presentation to JSP supported by MSS’s webmanager software with business logic implemented as Java EJBs. For the batch system, freestanding Java programs, each running in its own JVM, were generated.

Migrate!COBOL was used to convert the Cobol system to Java for compatibility with the migrated Linc. The resulting code was virtually line-for-line with none of the ‘bloat’ usually associated with such conversions. The new code is structured so as to be easily understood and maintained by the original programmers with minimal retraining in java, and has been well-received by the developers.

Both the Linc and Cobol systems were moderate scale - the Linc system consisted of 120 ispecs (screen layouts) and 60 reports (batch programs) and the Cobol system totaled 100,000 lines of code. Additionally 120 WFL job control programs were migrated, using
migrate!WFL, to shell scripts and these were controlled by a schedule run by standard system software (cron).

CHALLENGES

The Cobol system was the first live implementation of the Java migration path for the migrate!COBOL software and required significant R&D during the project. In spite of this the project was completed in a timely manner in a 9 month timeframe which included extensive testing.

The database definition and data had to be migrated to a format consistent with TPS internal standards for database usage. This was done, in the main, by enhancing the existing software and providing generated schemas that, functionally, fully replace the original DMSII.

During the project, the stringent security requirements of the Police Service had to be respected, and the core migration was done remotely in a “hands-off” manner. The team evolved ways of working that complemented the TPS project staff and made the task as efficient as possible.

RESULTS

The migration project was completed in 9 months and went live in May 2008. The system ran successfully from day one with improved performance and Toronto Police are very pleased with the result.

Using the well-proven migrate!LINC product and the newer migrate!COBOL (Java) technology, the actual migration was accomplished in 3 months and, after extensive testing, went live at the beginning of May 2008.

This is an achievement considering the changes involved: Linc migrated to Java, Cobol migrated to Java, WFL to shell script, scheduling system to crontab, DMSII database to DB2, COMS replaced by Websphere and the underlying MCP platform replaced with a standard AIX (IBM p Series) system.

Not only are the end-users benefitting from a web browser interface and the operations department having a standard environment to manage, but the developers also have a better environment to work in. A range of development technologies are now available as well as all the advantages of programming in Java. The transition was also eased by the layout of the new code being as close as possible to the original - even in the case of the automated COBOL to Java conversion where a virtual 1:1 ratio of new to old code lines was achieved.