

The COBOL Transformation Toolkit: CORECT

Technical Analysis

The 3 Steps to renew your legacy COBOL Applications

- 1 Document COBOL Applications
- 2 Extract Business Rules and Process
- 3 Transform to maintainable JAVA, C#, or MF COBOL

www.softwaremining.com





Contents

- OVERVIEW 3
- RE-HOSTING AND TRANSLATE TO JAVA / C# AT THE SAME TIME! 3
- ANALYSIS 4
- DOCUMENTATION 5
- BUSINESS RULE EXTRACTION..... 6
- HANDLING OF COBOL FILE DESCRIPTORS 7
- TRANSLATION TO JAVA, C# OR MF COBOL 8
- DATA SHEET..... 9
- FREQUENTLY ASKED QUESTIONS..... 10
 - Support For External COBOL Libraries* 10
 - Migration of Existing Data* 10
 - Supported Dialects* 10
- SERVICE DESCRIPTION..... 10
 - Free Evaluation* 10



Overview

COBOL Transformation Toolkit (CORECT) is designed to allow organisations move away from the restrictions and inherent costs associated with COBOL applications yet retain their investment in their legacy systems.

SoftwareMining's CORECT Tool facilitates the transformation of COBOL application to a variety of languages such as Java (J2EE), C#/ASP and even ANSI/MF COBOL.

The Primary Objective of CORECT has always been to generate 'legible and maintainable' code. The quality of code substantially affects the costs associated with future maintenance of the application.

The migration/transformation process can be achieved via 2 routes:

1. Heuristic based translation – where the AI based translator applies its own set of simplification rules to achieve legible and maintainable code.
2. Application of Business Rule Extraction on top of the heuristic based translation for more detailed customizations.

This then unlocks the legacy application making it ready for a total transformation service, where it is re-written in a variety of languages to suit the client requirement including Java (J2EE, Tomcat and WebSphere application servers), C# (For Microsoft .NET platform), MF COBOL, and Borland Delphi / Kylix (Linux Platform).

Approximately 95% of the legacy code is translated automatically using SoftwareMining's proprietary Artificial Intelligence-based tools with the remainder being dealt with in a semi-automatic manner by specialist transformation teams. This combination of innovative, unique, leading edge tools with specialist human intervention results in robust, high quality, flexible, scalable and fully documented code with a timescale of a few days rather than years for a fully manual process (e.g. 100,000 lines of COBOL code would take 1-2 man years to manually translate the SoftwareMining method takes between 10 and 20 days). In addition, during the transformation, the dead code and data groups are identified and such sections are commented out.

Re-hosting and Translate to Java / C# at the same time!

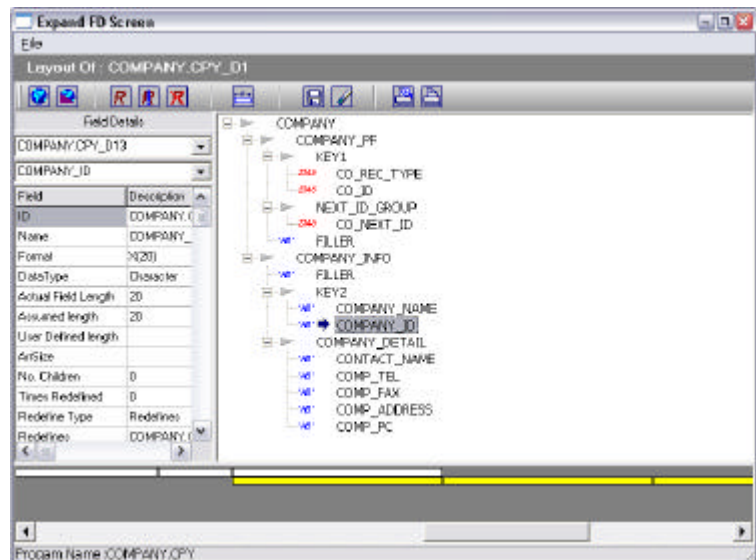
After the analysis of an application, CORECT can generate MF COBOL (re-hosting code) *and* Java/C# code simultaneously i.e. whilst addressing the immediate re-hosting issues – the system can also produce Java or C# together with the Oracle SQL DDL code.

This opens the door for strategic planning re the future of the application: how feasible is migration into a modern language/platform (Java, C# Oracle), what are the associated costs, what level of training is required, what is the Return-On-Investment, is the new code maintainable.

Analysis

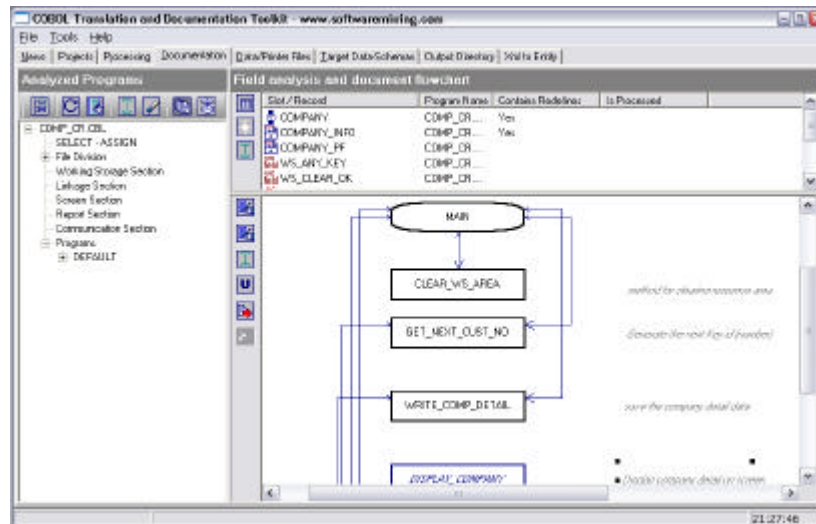
COBOL Transformation Toolkit can read most COBOL dialects. During the analysis phase, the system constructs internal models for both data structures and program logic. The analyzer uses heuristic based algorithms to identify data usage, identify and remove dead code, simplify the REDEFINITIONS, restructure into procedural code into an Event Driven model and more.

Identification and handling of REDEFINITIONS are done automatically via system heuristics. More demanding definitions can be handled via inbuilt, easy to use REDEFINITION identification and re-mapping screens.



Documentation

The CORECT documentation module operates as an extension to the Analyser, generating documents and reports on its findings. The system uses UML Activity diagrams for representation of process flow-logic, and generic HTML documentation for representation of Hierarchical structures of the COBOL File definitions.



The system offers the following features:

- Documentation of process logic in the form of UML Activity Diagrams
- Documentation of hierarchical File Definitions in HTML
- Identification of record-usage / us-used records within each program
- Identification of dead code
- Processing of Logic Documentation at 2 levels:
 - Interaction between program sections/labels
 - Application wide interaction (how programs interact)
- Ability to add manual notes / enhancements to the Activity Diagrams
- Export of the Diagrams to SVG/HTML for general viewing

Business Rule Extraction

During the transformation phase, CORECT uses heuristics to remove dead code, remove unused variables and to generate legible and well-constructed code. However, sometimes the heuristic transformation is not enough and individual business rules need to be identified and extracted from the original code.

CORECT's Business Extraction Module is designed to work on top of the existing heuristics approach – to provide further mining and classification of business rules. Mining of the business rules in CORECT can be achieved via:

- Filtering Statement Categories.
- Identifying operations on Data Items.

For example, the system can show non-persistence and non-screen handling operations on "Client" record within the "Order Processing" program. These filters identify a set of statements, which may include rules such as "how the customer is validated", "how customer new balance is calculated", "how the new invoice is generated" and etc.

The system provides functionality to further isolate the logical operations required in each rule and provides various means of saving the data.

- The operations comprising a business rule are written out as code snippets in Java, C#. An OWL, as well as MDA support, is currently being engineered into the module.
- The code snippets can then be exported into pre-defined 'Templates'.

The screenshot displays the 'Business Rule Extraction - COMP_CR.CBL' application. It features a 'Program Struct' pane on the left showing a tree view of methods: GET_NEXT_CUST_NO (with sub-items Mah, If, If, Mah, Mah, Mah, Mah, Mah), WRITE_COMP_DETAIL, and GET_COMP_DETAIL (with sub-item Display). The 'Generated Code' pane on the right shows a Java method signature: `public void Get_Next_Cust_No () throws Exception {`. The code includes comments like `// - Potential Bug - START statement containing 1 FD with multip` and `// - One of the following findBy... statements will probably be`. It also shows `Company_Pf.setKey1(0);`, `Company_Info.findByKey1("");`, `if (Company_Info.eof())`, `Company_Pf.setCo_Next_Id(0);`, `Company_Pf.findByKey1("");`, and `if (Company_Pf.eof())`. Below the code are three tables: 'Data Items' with columns 'Slot / Record', 'Program Name', and 'Contains...'; 'Data Filter' with columns 'Slot / Record', 'Program Name', and 'Contains Redefines'; and a 'Statement Filter' pane with a legend for statement types: Exit statements (green), File operations (red), Persistence statements (blue), Embedded SQL statements (yellow), Screen statements (magenta), Perform statements (cyan), and Call statements (purple).

Handling of COBOL File Descriptors

CORECT, by default, converts COBOL FD structures to (ANSI) SQL tables.

These tables are represented in code (Java, C#...) by their own access classes - called Data-Wrappers Beans. The data wrapper beans - are similar in functional objectives to EJB Entity Beans. They are responsible for their own persistence, SQL, getters and setters.

These Data-Wrapper classes can be implemented as EJB Entity beans, EJB Session beans (Default for J2EE Solution), local JDBC classes (default for client/server deployment), CORBA components or fit into any other architecture appropriate to the application.

The Data-Wrapper classes also contain all the relevant SQL code internally. This implies that the underlying persistence implementation can change without any changes to the main Business Logic.

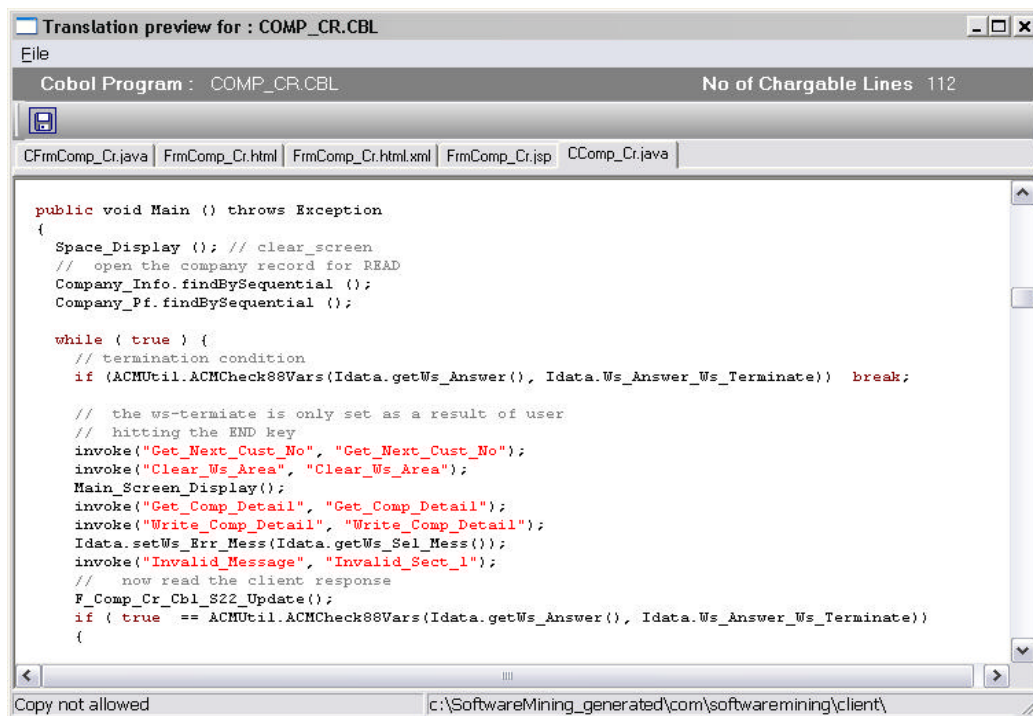
In fact - since the Data Wrapper beans are responsible for their persistence - and the application business logic does need any knowledge of their internals, the underlying implementation can be anything: **ORACLE, DB2, object databases, XML databases, even wrappers around back end COBOL applications or anything else.**

Each SQL table is represented within a separate (Data-Wrapper) class. This approach keeps the Data-Wrappers close to EJB Entity Beans - these classes take care of their own persistence and the application business logic will not contain any SQL. The Data-Wrapper classes can be implemented in a variety of architectures such as JDBC Wrappers, Entity Beans or EJB Session Beans. \The generated code can be deployed by different means. The data-wrappers can be re-generated to meet different deployment strategies.

In an approach following the OMG's Model Driven Architecture (MDA), CORECT generates Platform Independent Models (PIM) from COBOL File Definitions. These can be passed into different code generators to generate Platform Specific Model / Implementation. Therefore the underlying persistence (database), communication means (J2EE, CORBA or native), and the implementation (e.g. J2EE Entity bean or J2EE Session Bean) is future proofed, and can be altered at a future date – without affecting the business logic.

Translation to Java, C# or MF COBOL

CORECT Translation Toolkit can generate code for a variety of development languages – Java / J2EE, C# / .NET, Borland Delphi, or Borland Kylix for a pure Linux deployment.



```
public void Main () throws Exception
{
    Space_Display (); // clear screen
    // open the company record for READ
    Company_Info.findBySequential ();
    Company_Pf.findBySequential ();

    while ( true ) {
        // termination condition
        if (ACMUtil.ACMCheck88Vars(Idata.getWs_Answer(), Idata.Ws_Answer_Ws_Terminate)) break;

        // the ws-terminate is only set as a result of user
        // hitting the END key
        invoke("Get_Next_Cust_No", "Get_Next_Cust_No");
        invoke("Clear_Ws_Area", "Clear_Ws_Area");
        Main_Screen_Display();
        invoke("Get_Comp_Detail", "Get_Comp_Detail");
        invoke("Write_Comp_Detail", "Write_Comp_Detail");
        Idata.setWs_Err_Mess(Idata.getWs_Sel_Mess());
        invoke("Invalid_Message", "Invalid_Sect_1");
        // now read the client response
        F_Comp_Cr_Cbl_S22_Update();
        if ( true == ACMUtil.ACMCheck88Vars(Idata.getWs_Answer(), Idata.Ws_Answer_Ws_Terminate))
        {

```

The primary objective of Softwaremining is to reproduce legible and maintainable code. The Translation Toolkit maintains original comments (where possible), and uses Object Oriented Entity design to simplify the generated code

The generated code fits into a transparent framework of libraries, which facilitates the Communication, Deployments and Screen Handling. For example, via a simple change of underlying framework – the application can change from a working in Java client/server mode (preferred means of development) – to a JSP/J2EE mode (preferred mode of deployment), or the communication means changed from local to distributed architecture.

Our framework strategy helps in future-proofing the generated code by separating the non-functional aspect of code (communication, persistence, etc) from the actual business logic. The non-functional areas can then be upgraded without modifications to the business logic – e.g. the system can move from pure EJB to Web-Services/EJB architecture by simply changing the appropriate layer. This is achieved via incorporating the latest Object Oriented programming and design patterns in the generated code and the underlying framework.

The generated code separates the non-functional aspect of code (communication, persistence, etc) from the actual business logic. The non-functional areas can then be upgraded without modifications to the business logic – e.g. the system can move from pure EJB to Web-Services/EJB architecture by simply changing the appropriate layer.

Data Sheet

COBOL Analysis:

- Automatic "best fit" searches for REDEFINED variables
- Automatic conversion of PROCEDURAL to EVENT DRIVEN Code

Code Generation:

- Identifies dead code (unused methods), and unused record structures.
- Break down of COBOL programs into smaller Java / C# Classes. This is achieved by:
 - Separation of Working Storage variables to separate Java Data-Containers (serialized)
 - Separation of File Descriptor into persistent data-container (can sit on top of J2EE Entity beans)
 - Separation of Screens into separate Java Bean Classes (along with the associated JSP)
 - Assist in breakdown of large programs via business rule extraction (see later), or break down by visualization of PARAGRAPH/ SECTION / LABELS
 - Configurable framework: can change deployment from JDBC to EJB, or from JSP to AWT/SWING by changing the underlying libraries. This can improve the 'developer' performance at build and test times
- Translates COBOL FD to SQL (Oracle, MSSQL, MySQL or DB2 or ANSI)

Documentation:

Documentation produces Flowcharts / Activity diagrams for:

- Method (COBOL section / label / paragraph) interaction
- Inter-Program interaction
- The Documentation Default Template is HTML+ SVG

Business Rule / Business Process Extraction

- Extraction of Business processes by statement category (e.g. don't show screens manipulation or persistence statements)
- Isolation of Business Rules-Process associated to a Record/Variable
- Export of the rules into user definable Template: e.g. Java programs, C# programs, JSP programs, HTML documentation, and etc

Code Obfuscation:

We appreciate that there may be times that specific expertise of SoftwareMining support team could prove beneficial in translating a 'difficult' program. We also appreciate that the program code may contain sensitive information – and there will be reluctance to send the code off-site. In these circumstances CORECT's built in obfuscator can be used to remove all intelligible material from that code, and opens the door to SoftwareMining support team to provide their expertise in finding the best approach to the problem.

Frequently Asked Questions

Support For External COBOL Libraries

Embedded SQL – is regenerated into SQLJ or Non-SQLJ architecture.

CICS – A large subset is handled.

Migration of Existing Data

Data-migration services offered by SoftwareMining generate COBOL programs that export the data to a format ready to be populated into the new SQL structures.

Supported Dialects

SoftwareMining's COBOL Transformation Toolkit can read most COBOL dialects. Whenever we come (or our clients report) a set of new syntaxes – then support will be built into the next release.

Support for other languages such as Fortran and Basic is planned for the future versions of the product.

Service Description

SoftwareMining can offer transformation services at various levels:

1. Analysis and Transformation: SoftwareMining offers its wealth of experience in transformation projects to undertake the Analysis and Transformation of the code. This is the most demanding phase within the whole project, and our experiences could help in better identification of business rules, and a better result in transformation. The new generated code is then passed on to the client – for build and deployment. This service can offer a range of software testing, at one end of scale we deliver un-compiled code and at the other end of scale fully compiled, unit tested code can be delivered.
2. Turnkey Service: In addition to the Analysis and Transformation services above, SoftwareMining can undertake the entire project – delivery of a fully working, tested application, with no disruption to clients' existing business

Free Evaluation

To facilitate the evaluation of SoftwareMining's unique solutions and to prove the concept and value of the Transformation Process, a free translation into Java of up to 10,000 lines of code including the COPY BOOKS is offered.

Please contact info@softwaremining.com for further details/ NDA etc.