

COBOL Modernization

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#

Contents

| | |
|----------------------------------------------------------------------------------|----|
| OVERVIEW..... | 3 |
| ANALYSIS..... | 4 |
| DOCUMENTATION | 5 |
| BUSINESS RULE EXTRACTION | 6 |
| AUTOMATIC RE-ARCHITECTURE, RE-DESIGN AND RE-WRITING COBOL CODE JAVA AND C# | 7 |
| <i>Object-Relational Data Access</i> | 7 |
| <i>Object-Relational Data Access</i> | 8 |
| DATA-MIGRATION | 9 |
| DATA SHEET | 10 |
| SERVICE DESCRIPTION | 11 |
| EVALUATION | 11 |

Overview

The COBOL to Java transformation capability 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.

COBOL applications can be migrated to a variety of languages, such as Java (J2EE) and C#/ASP.

The Primary Objective 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 languages to suit the client requirement including Java (J2EE, Tomcat and WebSphere application servers) and C# (For Microsoft .NET platform).

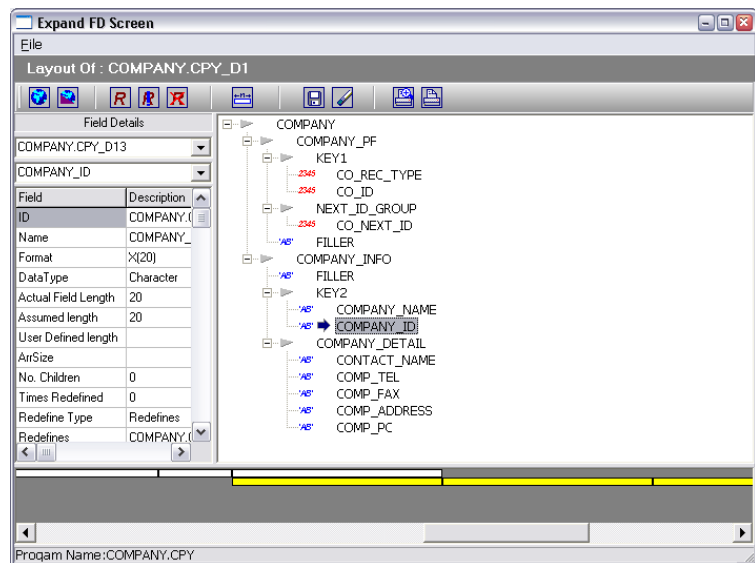
Approximately 98% of the legacy code is translated automatically using proprietary Artificial Intelligence-based tools with the remainder being dealt with in a semi-automatic manner by specialist transformation teams. This combination of innovative tools with specialist human intervention results in robust, and fully documented code with a timescale of a few days rather than years for a fully manual process.

For a comparison, 100,000 lines of COBOL code would take 1-2 man years to manually rewrite, but the highly automated method takes less than 1 week). In addition, during the transformation, the dead code and data groups are identified and such sections are commented out.

Analysis

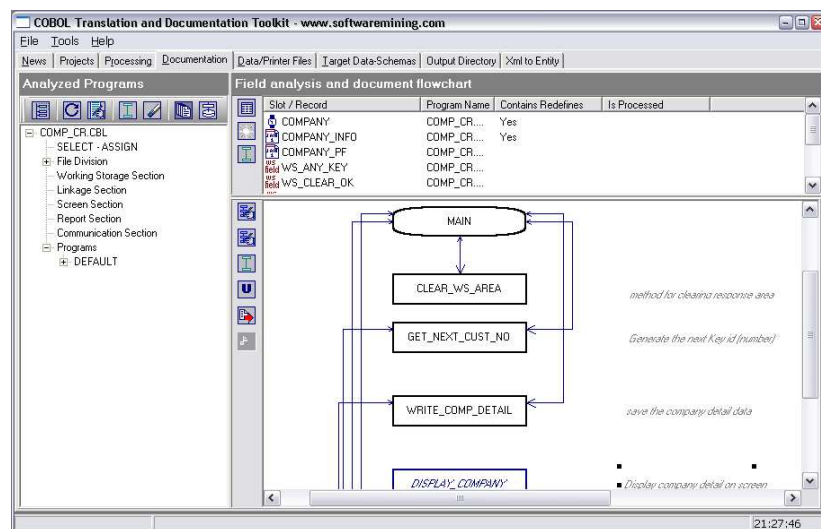
COBOL Transformation Toolkit can read most COBOL dialects. During the analysis phase, the system constructs internal models for both data structures and program logic. The analyzer uses heuristic based algorithms to identify data usage, identify and remove dead code, simplify the REDEFINITIONS, restructure 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 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, heuristics are used 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.

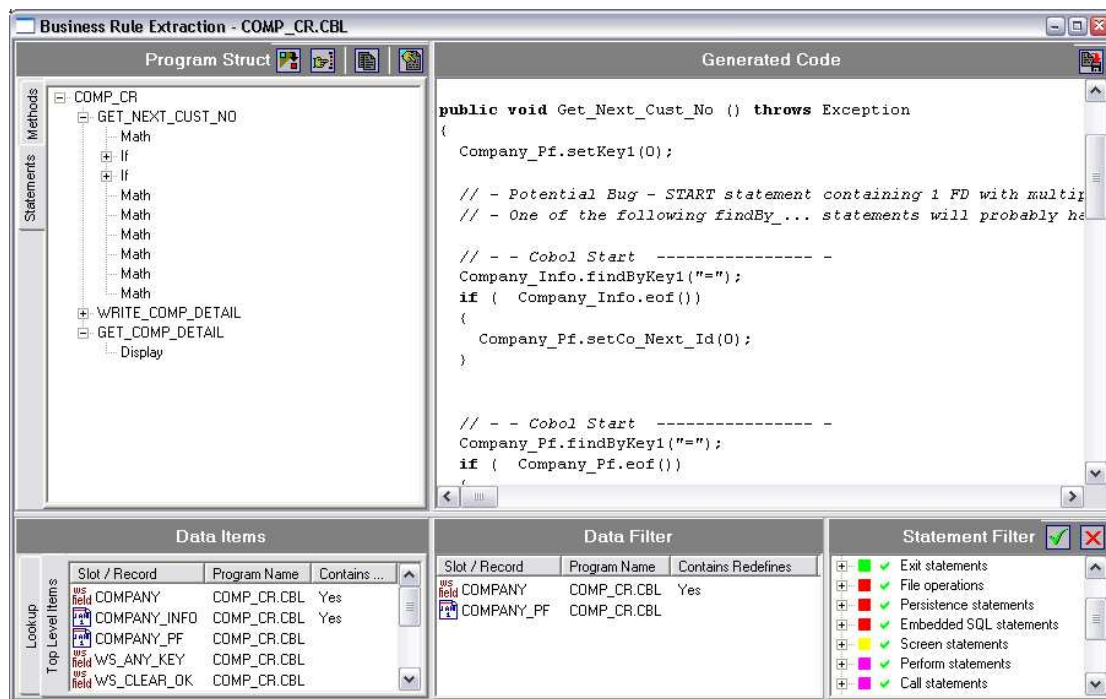
The Business Extraction Module is designed to work on top of the existing heuristics approach – to provide further mining and classification of business rules. Mining of the business rules can be achieved via:

- 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’.



Business Rule Extraction - COMP_CR.CBL

Program Struct

- COMP_CR
 - GET_NEXT_CUST_NO
 - Math
 - If
 - If
 - Math
 - Math
 - Math
 - Math
 - Math
 - Math
 - Math
 - WRITE_COMP_DETAIL
 - GET_COMP_DETAIL
 - Display

Generated Code

```
public void Get_Next_Cust_No () throws Exception
{
    Company_Pf.setKey1(0);

    // - Potential Bug - START statement containing 1 FD with multipl
    // - One of the following findBy... statements will probably ha

    // - - Cobol Start -----
    Company_Info.findByKey1("");
    if ( Company_Info.eof())
    {
        Company_Pf.setCo_Next_Id(0);
    }

    // - - Cobol Start -----
    Company_Pf.findByKey1("");
    if ( Company_Pf.eof())
    {
    }
}
```

Data Items

| Slot / Record | Program Name | Contains ... |
|---------------|--------------|--------------|
| COMPANY | COMP_CR.CBL | Yes |
| COMPANY_INFO | COMP_CR.CBL | Yes |
| COMPANY_PF | COMP_CR.CBL | |
| WS_ANY_KEY | COMP_CR.CBL | |
| WS_CLEAR_OK | COMP_CR.CBL | |

Data Filter

| Slot / Record | Program Name | Contains Redefines |
|---------------|--------------|--------------------|
| COMPANY | COMP_CR.CBL | Yes |
| COMPANY_PF | COMP_CR.CBL | |

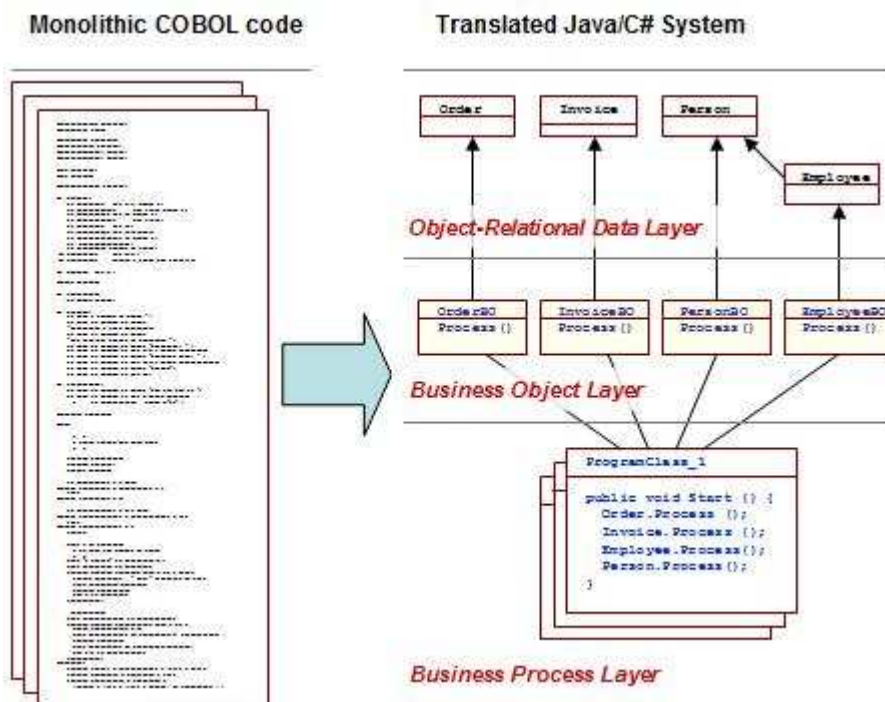
Statement Filter

- Exit statements
- File operations
- Persistence statements
- Embedded SQL statements
- Screen statements
- Perform statements
- Call statements

Automatic Re-Architecture, Re-Design and Re-writing COBOL code Java and C#

The primary objective is to reproduce legible and maintainable code. The Application Modernization Toolkit uses Artificial Intelligence techniques to break down large monolithic COBOL programs into a re-designed, layered Object Oriented system featuring:

- Inheritance driven Data layer (e.g. Employee class Inherits from Person Class)
- Business Object Layer (e.g. EmployeeBusinessObject containing methods for calculation of Employee Salary)
- Business Process Layer - orchestrating interactions between BusinessObjects



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.

The 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.

Object-Relational Data Access

By default we convert 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 OMG's Model Driven Architecture (MDA), We generate 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.

Data-Migration

The translation of VSAM file definitions need to be granular, normalized, and use separate columns for representation of each VSAM field. The migration also needs to account for COBOL REDEFINES keyword, Group access, as well as COMPUTATIONAL values. More importantly this activity needs to be done FAST to provide a good RoI (Return on Investment) and not to delay the new developments.

In our unique approach - we address all these issues, and go further by providing specialist Object-Relational Java or C# data-access layer and XML integration layer. These features enable the new developments to start and tested very quickly.

Special Features

- Source platforms: VSAM, QSAM and ISAM, Unisys DMS-II, IMS, IDMS
- Support for Cobol REDEFINES, and COMPUTATIONAL values
- Target platforms: ANSI SQL, DB2, Oracle, MS-SQL Server or MySQL
- Java and C# Object-Relational Mapping Data Access layer.
- Multiple Java Deployment strategies: J2EE or pure JDBC
- Includes auto-generated Java/C# data Object-Relational Data Access Layers

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

Service Description

TSG can offer transformation services at various levels:

1. **Analysis and Transformation:** TSG 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, we can undertake the entire project – delivery of a fully working, tested application, with no disruption to clients' existing business

Evaluation

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

This document was created with Win2PDF available at <http://www.win2pdf.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.