International Journal of Computer Applications (0975 – 8887)
Volume 79 – No5, October 2013

Design of a Reverse Engineering Model (A Case Study of COBOL to Java Migration)

Aditya Trivedi
School of Computer Science & IT- DAVV
Devi Ahilya Vishwavidyalaya, Indore

Ugrasen Suman
School of Computer Science & IT- DAVV
Devi Ahilya Vishwavidyalaya, Indore

ABSTRACT

With the progress of the software technology, the existing legacy systems are becoming obsolete and unable to satisfy the customer needs and expectations. Most of the legacy systems designed using COBOL, as it is a programming language. On the other hand, today Java widely used programming language for designing systems. The Java is pure object-oriented, where as the COBOL is procedure oriented programming language. The legacy systems designed earlier needs the huge amount of maintenance. The programmers of these legacy systems are now getting old moving into the retirement. After that, for people maintaining legacy systems it will be more difficult because keep up these legacy systems needs expertise in the programming language. Therefore, it is necessary to propose a framework to migrate existing legacy COBOL systems to object-oriented Java platform. The advantages of this migration process are that the upholding of the system running in the different organizations will be easier than the legacy COBOL systems.

General Terms

Reverse Engineering, Forward Engineering, Reengineering, Legacy Systems, XML, XML Schema Definition,

Keywords

Legacy code to Object Oriented code, COBOL to Java, Migration of COBOL to Java, Legacy System Migration.

1. INTRODUCTION

Nowadays, several government departments, private institutes, firms etc. are getting computerized and digital. If, we look few years back with the context of developing those software systems on which these departments and firms process their data based on some programming language. In fact, all the software systems today based on programming languages, but due to continuous evolution of programming languages are in trend. Programming languages and constructs are moving towards the continuous improvement in their features. The software system used these programming languages are specific to a user. Many software systems used procedural programming language (COBOL, PASCAL and FORTRAN etc). The scenario has changed now a day and the new programming constructs in trend (object-oriented programming). This research, based on the reverse engineering of the legacy systems to the object oriented system. The reason behind this migration is, shifting from old technology to new technology, by the programmers. So, the old programmers and the programming languages are getting obsolete. Thus, the softwares based on these procedural languages are difficult to keep up. But, these softwares not turned out. This software's do consistently and efficiently. The large amount of data stored and processed by these softwares. Thus, the migration needed of the data and the software from one structure to another. This research is just to give an architectural approach to migrate in the context of COBOL systems to Java through XML. Java is an easy and yet prevailing object-oriented programming language. It developed to give platform-independency.

We are proposing an approach, which convert COBOL program to XML schema and this equivalent schema converted to equivalent UML diagrams. Finally, this diagrams used in forward engineering process and generated Java classes as a target language. XML schema chosen as the intermediate result because XML is flexible and compatible with other software and hardware environment. Ultimately, this scheme first, reverse engineers the COBOL code and finds the domain model. Now, forward engineering process applied on domain model to generate equivalent target programming language. One of the difficulties associated with this software re-engineering projects is that the program source code invariably stored in plain text format. This format does not imitate the fundamental structure of the program. So, the software re-engineering or code migration tools needed to discover this structure. This paper sees the sights of the prospects of accepting XML format to represent program structure for software systems.

The growth of the object-oriented languages and the component technologies, give more robust and maintainable software systems. On the other hand, the decline in the number of programmers fluent in languages such as PL/IX, which make it highly desirable to migrate the legacy software systems. Thus, their need a tool, which migrates. The fact, program source code typically stored as a plain text, means the structure of the code hidden. The specialized tools required to show this structure. XML is an ideal format to represent program structure. There is a wealth of XML related tools to aid in visualization, analysis and transformation of code.

Most people refer to the Unified Modelling Language as UML. The UML is an international industry standard graphical notation for describing software analysis and designs. In UML, U stands for unified because UML is a standardization and unification of available modelling notations of Booch, Jacobson, Rumbaugh, among others.

Re-engineering is the process to identify the domain models from one source code and redesign the system into target code. For example, legacy systems migrated to new object-oriented systems. It is the combination of the both process (reverse engineering and forward engineering) with a little change of Δ in the identified domain model, from the source code. Reverse engineering is the process to generate the domain model from the source code. The forward engineering is the process to redesign the target system from this domain models.

2. RELATED WORK

There are several migration projects working in this area for migrating COBOL source code to Java. In year 2010, an industrial report on migration project often airport management system [1] presented. A set of tools described which used to transform a million lines of COBOL code into Java. Remain disadvantage of this migration project is the size of the resulting Java code amounted to three times as much as the original COBOL code. This explained by the following facts.

In generating the Java code, each COBOL procedural statement mapped to one or more Java statements. Thus, alone from the procedural code there is an increase of circa 50% in the number of statements. For instance the GOTO translated to an Assign and a Return. On the data side each data declaration converted to a set and a get method on the character array of the static object. Each such method has three statements. – An Entry, an Assign and a Return. So, there are three Java statements for each COBOL declaration. This expansion is partly compensated by fact that only the data items used accessed, but including the first assignment there are still 6 to 8 Java statements for every referenced COBOL data field.

The author proposes an approach for migration strategy through a toolkit implemented which automatically decompose the legacy software to find the software components encapsulated in different wrappers and creating new programs. The biggest problem met in re-engineering tool that we must carry out data flow analysis algorithms. The performances of the data flow analysis algorithms for identifying the interfaces of the new programs are satisfactory, also considering that the tool does not need any human interaction.[2]

The next work that will involve the toolkit for the next wrapping and incremental translation phases of the method. In fact, each identified object has encapsulated into an object wrapper. The wrapper will implements the connection through which newly developed objects access legacy resources. The long-term goal is the re-implementation of selected legacy object. Just, the objects whose reimplementation cost-effective replaced,, while the other legacy objects stay in use in their original form.

An automated COBOL to Java translation presented that allows for incremental migration and does not guarantee complete functional equivalence of the generated code with the first program [3]. In this article, an automated approach presented for source-to-source translation of COBOL applications into Java. We demonstrated conversion method, which aimed for producing more maintainable code in comparison with other approaches for language conversion. In some cases the programs produced with this approach are not functionally equivalent to the original ones, An algorithmic approach to reverse engineering of procedural systems to object-oriented platform presented which uses domain models of the procedural languages to represent it at a very higher level of abstraction in the form of abstract syntax tree[4]. The reverse engineering process starts with internal source code representation of the procedural system for system understanding and analysis. The internal representation based on XML as a portable source code representation that use a procedural language domain models. The abstract syntax tree used to find the structural relationships among data items. The syntactical structure of the language domain models map to the XML DTD. The procedure begins with identifying candidate classes by analyzing global data structures and function formal limits in a single source code file. The same procedure repeated for multiple source files in a project that results a complete object-oriented model.

A transformation process and tool C_2O^2 COBOL to Object-Oriented for analyzing COBOL applications presented [5]. The tool is capable of identifying classes and their relationships with a process of understanding and refinement in which COBOL data structures analyzed, aggregated, and simplified semi automatically. The tool can support different re-engineering methods. It used on a large application such as the software for managing all libraries of the University of Florence.

A framework is presented to migrate systems written in procedural code into an object-oriented platform [6]. The software analysis based on the portable XML source code representation that uses a domain model and so, maps the entities of the programming language domain model to a DTD. The XML document effectively corresponds to an AST and has a standard structure and API. Also consider that such an XML based representation of the source code thought of as first step towards CASE tool interoperability.

In this paper [7], overviews of research on reverse engineering XML schemas into UML diagrams. A conversion and transformation process presented. Converting a XML schema into a graphical form i.e. UML. UML constructs used to form a UML pattern are predefined in UML. This makes the resulting UML diagram simpler and generally understandable. The transformation process includes all building blocks of XML schema [8].

A method for reverse engineering XML documents based on three-level-model and uses a class diagram of UML [9]. Conceptual model maximizes understandably of semantic concepts by user's knowledge of UML, but not XML schema. The middle level model addresses the range of XML schema concepts, such that any XML schema can be expressed in UML.

3. PROPOSED WORK

We are proposing a model, which based on the reverse engineering process. This model transfers, source code (COBOL) to target code (Java). In this, XML used as a midway source code representation.

Our objectives are to define a reverse engineering model to migrate COBOL code to Java code, to map COBOL structure to XML schema/DTD, to map XML schema/DTD to equivalent domain model, to generate Java code through forward engineering.

There are several tools and strategies exist to transform COBOL source code into Java code. The problem with legacy COBOL systems is that after the five present five programmers go into retirement there will be no one left to keep up them. These problems associated with the fact that it is necessary to propose a framework to migrate legacy COBOL system to Java platform. Migration of COBOL or legacy system to Java platform is not new anymore. Several approaches proposed for migration. The code generated from these frameworks is three times larger than the code originally in legacy systems. In this project, we propose a re-engineering process framework for legacy COBOL system to Java code migration, with XML is as the intermediate language.

3.1 Proposed Framework

This research based on reverse engineering of COBOL source code and to identify the equivalent UML models. This UML models then converted into target Java code through forward engineering. The Figure 1 shows the proposed framework consists of following parts: COBOL file, COBOL to XML, XML to equivalent domain model, domain model to Java file. This discussed in the subsequent subsections.

The Figure 1 shows the model to migrate COBOL source code to target Java code. COBOL source files or COBOL copybook files are the first input to this model, towards COBOL to XML part. This COBOL to XML converts COBOL file to equivalent XML schema (XSD)/DTD (Document Type Definition). This XML schema definition file then transferred to generate its equivalent domain model, which generates the UML model for given XML schema. These two sub-processes COBOL to XML and the equivalent domain model logically treated as the part of the reverse engineering. The output of this reverse engineering process is the domain model equivalent to COBOL source files. This output from reverse engineering process then used as the input to generate the target Java code in the next phase, which logically treated as the forward engineering phase of the whole migration process. The code generated from this phase is the target code from COBOL source code. There are several migration models are available for migrating COBOL to Java. Figure 1 shows this process.

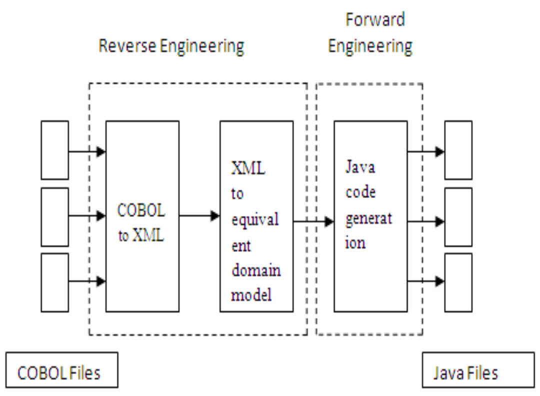


Fig 1: Framework for COBOL to Java migration

The re-engineering process is the collection of reverse engineering and forward engineering process, with change Δ in the artifacts of reverse engineering phase. Figure 2 shows the re-engineering process adopted in this project.

Conceptually, the COBOL procedural structure converted into XML tree, which in trn reverse engineered into domain model or UML model. This UML model, forward engineered into the object-oriented structure, which is the target object-oriented system. The process of re-engineering is the process to analyze a subject system and develop a completely new system.

3.2 Migration Process

There are four steps to migrate:

- Conversion of COBOL source code to XML.
- For XML, identified in the previous step, DTD/Schema produced. DTDs/Schemas used to validate the XML source.
- DTD/Schema then reverse engineered into the equivalent domain models.
- Forward engineering from domain model to the target Java code.

The Figure 3 shows the conceptual model for the framework. The diagram shows the process of reverse engineering and forward engineering. Thus, the transformation of the COBOL source code to domain model is the reverse engineering process. The remaining process, from domain model to Java target code is forward engineering.

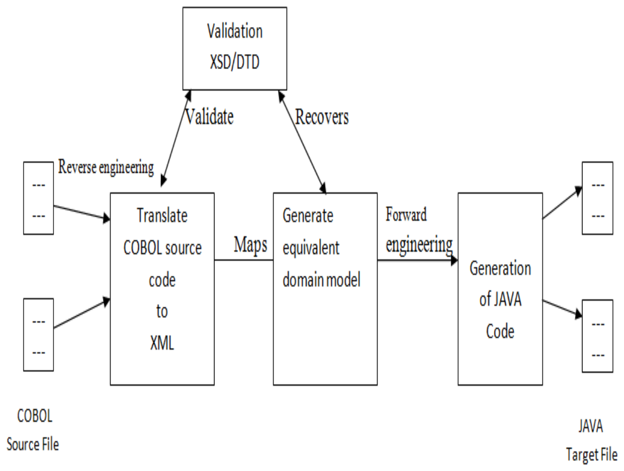


Fig 2: Conceptual Model for the Migration Process

3.3 Mapping XML schema to Domain Model

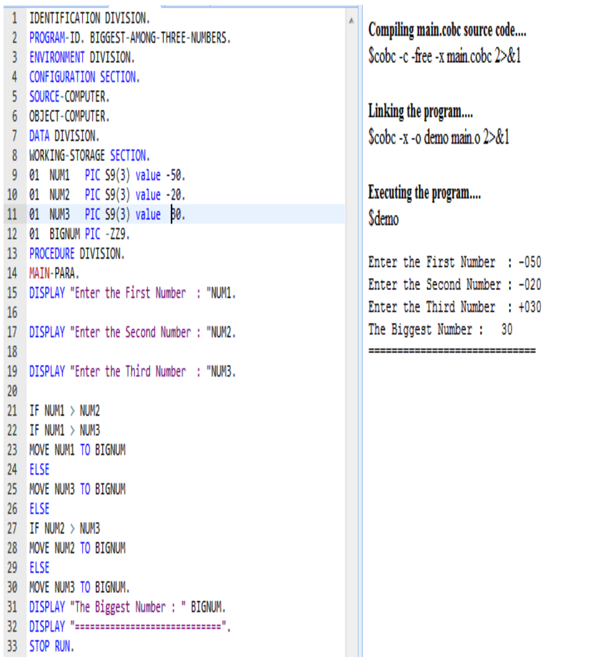
The mapping of XML Schema and domain model is done with the following domain model constructs; Table 1 shows the generic domain model constructs. The research on XML Schema to UML reverse engineering is less than the DTD to UML due to various reasons, such as the fact that DTDs have been used much longer than XML Schemas and thus most of the reverse engineering works have only concentrated on the conversion of DTDs to UML diagrams. Another reason may be that the reverse engineering target may not be UML diagrams but some other high level models.

Table1. Mapping of XML Schema to UML elements

XML Schema	UML Element
Element complex type, with ID attributes, and key	Class
Abstract element and complex type, with ID attribute	Abstract class
Subelement of class complex type	Attribute
Attribute of the corresponding element	Stereotype
Element without attributes	Package
Reference element with, IDREF attribute referencing the associated class and keyref for type safety (key/ keyref references)	Association, aggregation
Association class element and an additional IDREF references to association class element and a keyref in the corresponding reference elements in the associated classes	Association class
Extension of the reference element, keyref and key of target class with the qualified attributes	Qualified association
Reference element, with subordinate class elem. (hierarch. rel.)	Composition
Complex type of subclass is defined as an extension of the complex type of superclass	Generalization
Currently not mapped	Association constraint
Association element with IDREF references to all associated classes (resolution of the n-ary association)	n-ary association

4. IMPLEMENTATION AND RESULTS

On the basis of our experiments the COBOL program executed and results shown. Target Java classes generated for the COBOL program. COBOL program with result shown in snapshot 1.

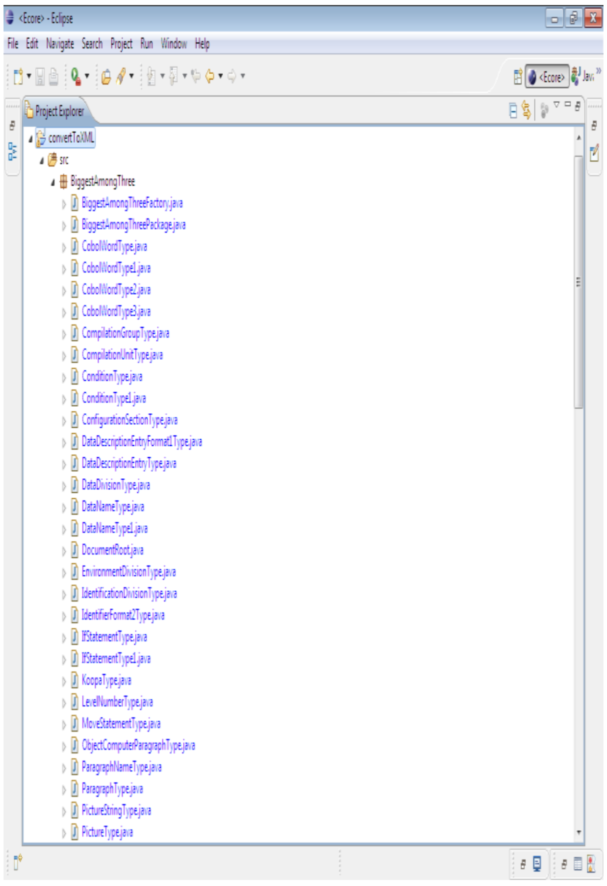


```
IDENTIFICATION DIVISION.
PROGRAM-ID. BIGGEST-AMONG-THREE-NUMBERS.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
OBJECT-COMPUTER.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  NUM1   PIC S9(3) value -50.
01  NUM2   PIC S9(3) value -20.
01  NUM3   PIC S9(3) value  30.
01  BIGNUM PIC -ZZ9.
PROCEDURE DIVISION.
MAIN-PARA.
DISPLAY "Enter the First Number  : "NUM1.

DISPLAY "Enter the Second Number : "NUM2.

DISPLAY "Enter the Third Number  : "NUM3.

IF NUM1 > NUM2
IF NUM1 > NUM3
MOVE NUM1 TO BIGNUM
ELSE
MOVE NUM3 TO BIGNUM
ELSE
IF NUM2 > NUM3
MOVE NUM2 TO BIGNUM
ELSE
MOVE NUM3 TO BIGNUM.
DISPLAY "The Biggest Number : " BIGNUM.
DISPLAY "============================".
STOP RUN.
```

```
Compiling main.cobc source code....
$cobc -c -free -x main.cobc 2>&1

Linking the program....
$cobc -x -o demo main.o 2>&1

Executing the program....
$demo

Enter the First Number  : -050
Enter the Second Number : -020
Enter the Third Number  : +030
The Biggest Number :   30
============================
```

Snapshot 1 COBOL file output

Java classes generated for the COBOL program shown in the snapshot 2. With the tool we used for experiments generated the target Java classes for domain models we provide as input.

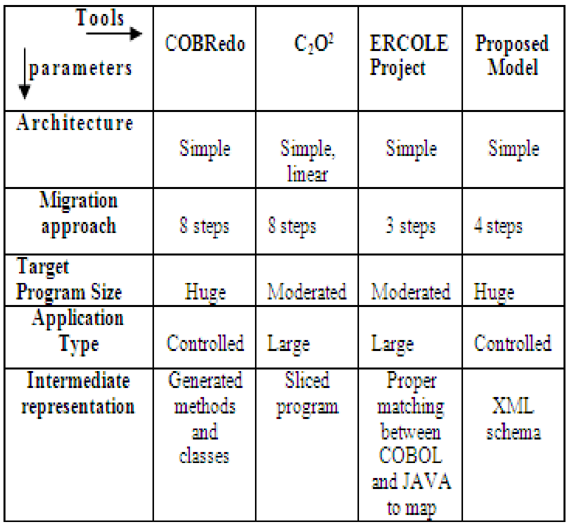


```
convertToXML
  src
    BiggestAmongThree
      BiggestAmongThreeFactory.java
      BiggestAmongThreePackage.java
      CobolWordType.java
      CobolWordType1.java
      CobolWordType2.java
      CobolWordType3.java
      CompilationGroupType.java
      CompilationUnitType.java
      ConditionType.java
      ConditionType1.java
      ConfigurationSectionType.java
      DataDescriptionEntryFormat1Type.java
      DataDescriptionEntryType.java
      DataDivisionType.java
      DataNameType.java
      DataNameType1.java
      DocumentRoot.java
      EnvironmentDivisionType.java
      IdentificationDivisionType.java
      IdentifierFormat2Type.java
      IfStatementType.java
      IfStatementType1.java
      KoopaType.java
      LevelNumberType.java
      MoveStatementType.java
      ObjectComputerParagraphType.java
      ParagraphNameType.java
      ParagraphType.java
      PictureStringType.java
      PictureType.java
```

Snapshot 2 Java classes

The proposed model is successfully generating the Java code from COBOL file. It can be extend for the large applications.

Table 2. Comparison with other tools



Tools → / parameters ↓	COBRedo	C_2O^2	ERCOLE Project	Proposed Model
Architecture	Simple	Simple, linear	Simple	Simple
Migration approach	8 steps	8 steps	3 steps	4 steps
Target Program Size	Huge	Moderated	Moderated	Huge
Application Type	Controlled	Large	Large	Controlled
Intermediate representation	Generated methods and classes	Sliced program	Proper matching between COBOL and JAVA to map	XML schema

5. CONCLUSION

In this project, re-engineering process activities examined, which uses various phases that aid in re-engineering project development. Also, we present a model for the reverse engineering of procedural platform into the object-oriented platform. We considered COBOL as a procedural language and Java as an object-oriented language. Starting with COBOL source code (internal source code), to represent the procedural system, for system analysis, in reverse engineering process. XML is portable, the internal representation given by XML source code that uses a procedural language domain models. This XML validated by the produced XML DTD/Schema. This XML Schema mapped to generate the equivalent domain model. The domain models based on the generic domain model constructs for XML schema to UML mapping. Forward engineering applied on this domain model to generate the target Java code. Thus, the procedural system re-engineered this way can improve the quality factors (maintainability, competence and flexibility).

6. REFERENCES

[1] Harry M. Sneed, “Migration from COBOL to Java: A Report from the field”, 26th IEEE International Conference on Software Maintenance in Timisoara, Romania, 2010.

[2] Carmine Albanese, Thierry Bodhuin, Enrico Guardabascio & Maria Tortorella, “A Toolkit for Applying a Migration Strategy: a Case Study”, Proceedings of the sixth European Conference on Software Maintenance and Reengineering, 2002 IEEE.

[3] Maxim Mossienko, “Automtated Cobol to Java Recycling”, Proceedings of the Seventh European Conference On Software Maintenance And Reengineering, 2003 IEEE.

[4] Dr. Ugrasen Suman & Dr. Maya Ingle, “Reverse Engineering of Procedural Systems: An Algorithmic Approach”, International Journal of Computer Science & Information Technology(IJCSIT), Serial Publications, NewDelhi, January-June 2008.

[5] A. Fantechi, P. Nesi, E.Somma, “Object Oriented Analysis of COBOL”. 1997 IEEE

[6] Ying Zou, Kostas Kontogiannis, “A Framework for Migrating Procedural Code to Object Oriented Platforms”.

[7] Augustian Yu, Robert Steele, “An Overview of Research on Reverse Engineering XML Schemas into UML Diagrams” Proceedings of the third international conference on information technology and applications(ICITA’05), 2005 IEEE.

[8] Flora Dilys Salim, Rosanne Price, Shonali Krishnaswamy, Maria Indrawan, “UML Documentation Support for XML Schema”, Proceedings of the 2004 Australian Software Engineering conference (ASWEC’04), 2004, IEEE.

[9] Yang Weidong, Gu Ning, Shi Baile, “Reverse Engineering XML”, Proceedings of the 2004 Australian Software Engineering conference (ASWEC’04), 2004, IEEE.

[10] Koopa COBOL Parser Generater.

[11] DTDGenerater “dtdgen 7.0”

[12] Eclipse JEE-JUNO

[13]Eclipse-Galileo Modelling Framework.