

## Code Restructuring and Software Improvement

We are going to discuss software modernization and code restructuring. Under code restructuring we mean changes in the code basis of a software system, such that in most of the cases the functionality of the software doesn't change but moving (relocating) code parts to different modules and packaging them in different packages and if necessary replacing some parts with more efficient code the efficiency, the effectiveness and the usability in new platforms rises. In our previous paper – Software Improvement through Model Transformation – we stated the main underlying techniques and reasons, which may make a revision inevitable. In this paper we want to go a little bit more into details and give an insight into the model operations and model manipulation features in the Eclipse modeling framework. As a CA Gen partner company we use a model transformation example from a PL1 program into a CA Gen model. The same transformation can also be made among any other two or more programming languages and related supporting infrastructural systems. For example the source system may consist of the COBOL and PL1 programs of an organization with any underlying TX monitor and database system.

Code Restructuring is sometimes necessary because of the technological changes since the program was developed initially. A block mode application which supports user interfaces in the so called 3270 terminal technology may not be suitable for SOA. The presentation layers and business logic layers of those programs are usually in one code block and the software service which only uses the business logic cannot be utilized without running the presentation logic part. The requirement to this sort of program coding resides in separating the business logic from the presentation logic, such that both can be called independently.

If the decision for restructuring is once taken, the next step is to decide which platform (software) to use for reverse engineering. Reverse engineering has various aspects. Those are syntactical reverse engineering and semantically reverse engineering. The time passed during the lifetime of a program may have left some traces. The code quality may not be as well as it was in the beginning of its lifetime. This may have an influence to the overall understandability of the program. The reverse engineering platform must be able to get the true functionality out of the system. This includes the correct interpretation of the code blocks. Some user interaction may be of advantage in the reverse engineering and restructuring process.

As we stated above the restructuring of a program may result out of many reasons. Building new modules in order to support a wider usability of the business logic may be a reason. Another one may be because of the need of conversion of the programming language into a better state-of-the-art one. For example if you want to make some change of paradigm as we do it in our case, the interaction of the modeler with the underlying platform is necessary. In our example we will take a 3270 PL/1 application and transfer the logic to CA Gen and to JAVA.

## Stages of Reverse Engineering

In the literature reverse software engineering is described as the process to find the hidden or difficult information about a software system. Reverse Engineering of a software aims at gaining back a state of the related software, which can be used to understand the logic and the rules behind the reverse engineered software. The reverse engineering may consist of one or many stages. The starting point of such an effort can be the machine code or as it is in our case some programs. In our case we use two stages. The first stage is the so called preprocessing. This stage must not be done for every source system. Most of the time the organization, which owns the reverse engineered software has its development strategy and depending to this strategy some sort of preprocessing can be done. If we take some PL/1 or COBOL sources and normalize those programs under consideration of some rules of the organization and make some modularization in order to support the following Reverse engineering steps.

### Preprocessing

We decided to preprocess the source code as the first step in the reverse engineering process. In this stage the code parts of four different categories (program source, copybooks, tables-declgens and screen maps for PL1 and COBOL) are inserted into a relational database and the program source is divided into modules. The utilization of a relational database versus the flat file structure is because of the programmatic accessibility of interest. In COBOL, we made modules from each SECTION in the Procedure Division and in PL/1 from each PROCEDURE. One advantage of this modularization is to give a preliminary feeling about the software construction and composition. If we would like to improve and reshape the CA gen Code we could use the Action Blocks as the modules. At this stage also some standardization is done such that the code looks nice and optically easier to understand. In figure 1 such a preprocessed PL1 transaction code is depicted. After the preprocessing stage the code can be investigated and the first ideas about the content can be gained. Some modules can be bound together to packages. Even some code changes can be made, since the preprocessed code is clear and lucid. Changes made in this stage require text editors.

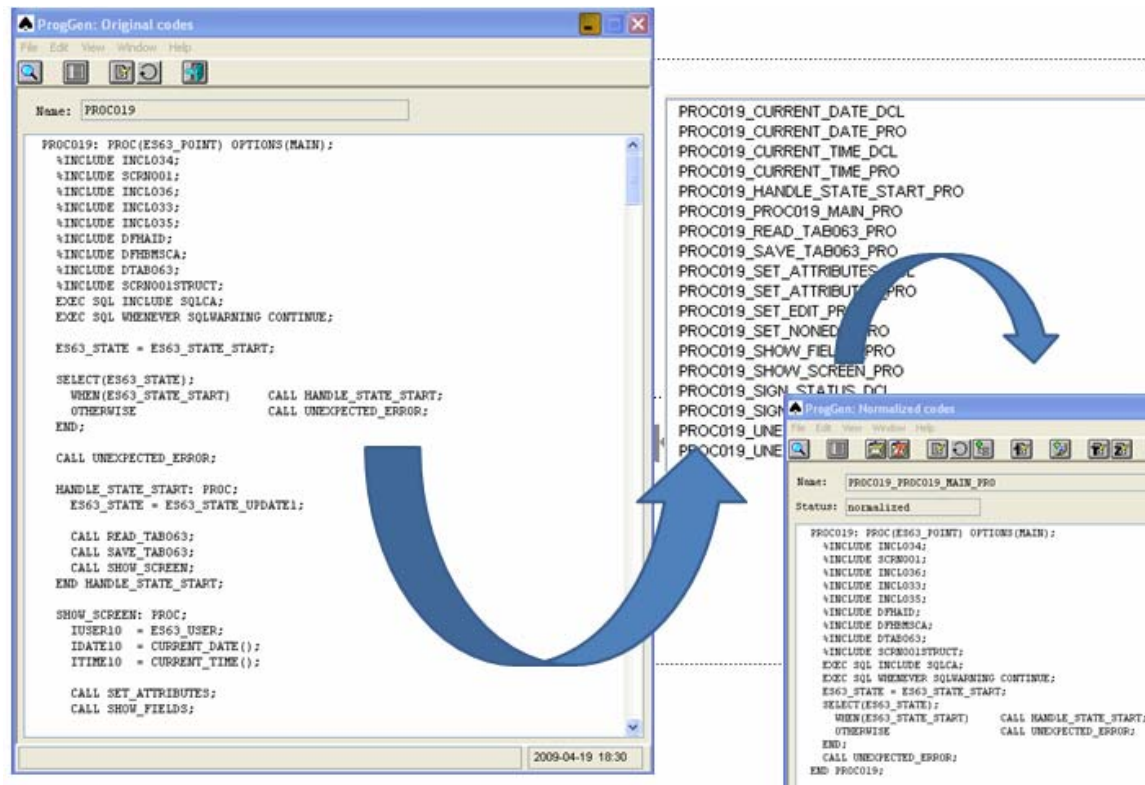


Figure 1

## Analyzer

The preprocessed software is easier to understand and also some restructuring may already be possible but for a final mining and understanding of the source code, we have to make more. We have to understand the whole source code syntactical and semantically. The software is written with a programming language like for example COBOL or PL/1. In order to understand the meaning of the code basis, we have to analyze it. Analysis requires parsing of the programming language. In computer science and linguistics parsing, or, more formally, syntactic analysis, is the process of analyzing a sequence of tokens (for example, words) to determine their grammatical structure with respect to a given (more or less) formal grammar. It means that we have to construct an analyzer which is able to analyze the program syntactical. We will describe the construction of the parser and the formal analysis of the language in another paper. Figure 2 depicts such a parser generator. Such parsers can be generated from the Backus Naur Form of a programming language or can be bought in the market. In computer science, Extended Backus-Naur Form (EBNF) is a metasyntax notation used to express context-free grammars: that is, a formal way to describe computer programming languages and other formal languages.

```

P sql_func
P sql_arithm_expr
P sql_days_arithm_expr
P sql_days_func
P sql_date_func
P sql_select

```

```

    ){actionHandler.endRule("sql_entityView");}
    ;
    sql_columnView
    :
    {actionHandler.startRule("sql_columnView");}
    {

```

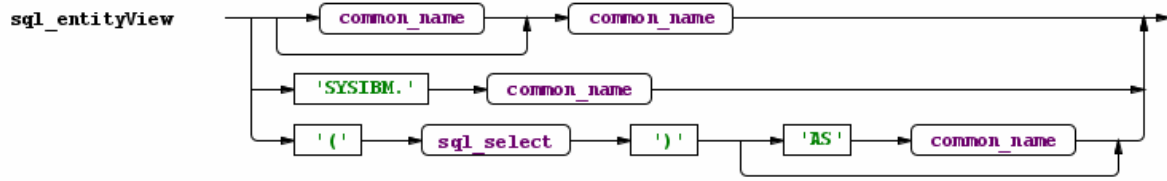
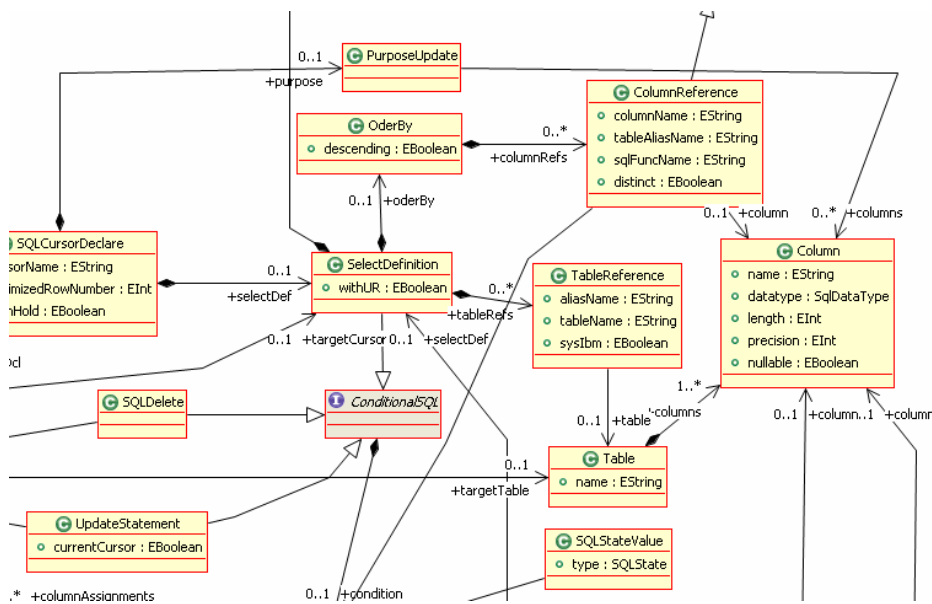


Figure 2

Semantic analysis is the phase in which the RE Tool adds semantic information to the parse tree and builds the metamodel instance. This phase performs semantic checks such as type checking (checking for type errors), or object binding (associating variable and function references with their definitions), or definite assignment (requiring all local variables to be initialized before use), rejecting incorrect programs or issuing warnings. Semantic analysis usually requires a complete parse tree, meaning that this phase logically follows the parsing phase, and logically precedes the code generation phase, though it is often possible to fold multiple phases into one pass over the code in implementation. Figure 3 below depicts a part of the metamodel after the semantic analysis.

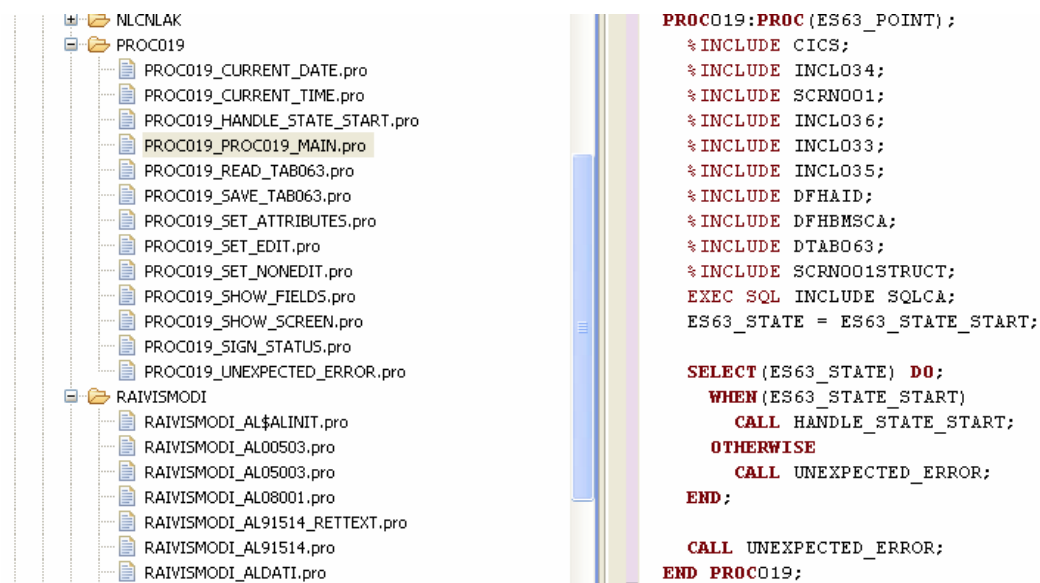


Now we have completed also the semantical analysis of the source. This means that we populated the metamodel instance and have now the ability to use the information we have in the metamodel instance programmatically. The question we have to answer is : Which platform would be suitable for the further work. Our further work is basically about model operations. Model search, model migration, model transformation and other model operations are necessary to finish the reverse engineering and the software restructuring. Therefore we decided to use Eclipse. EMF has many components we can use to fulfill our purpose. The seamless exchange of models among the environments in which we have the preprocessed

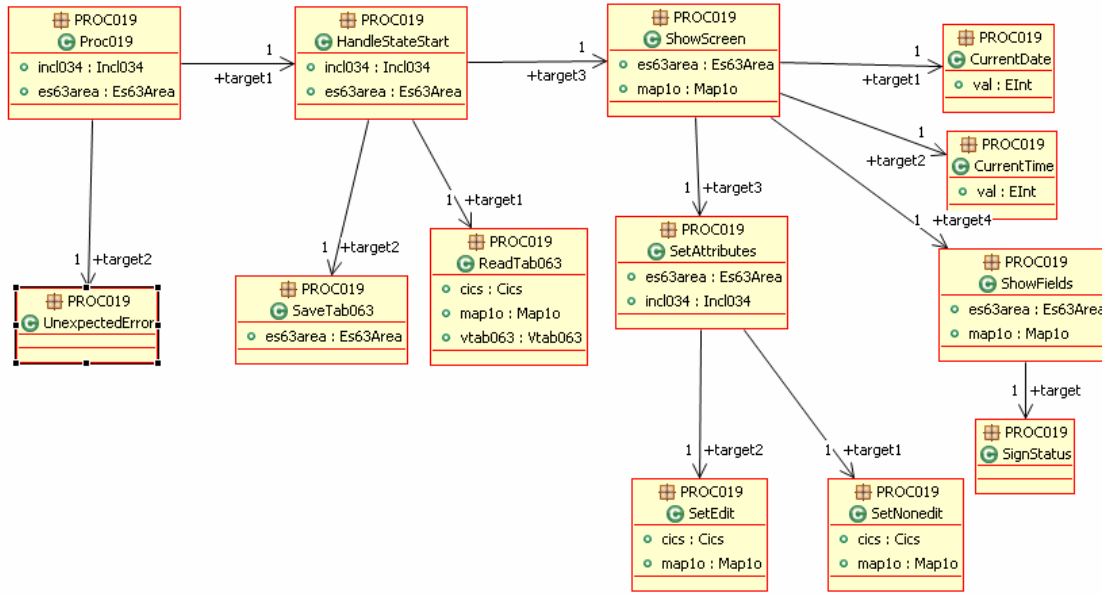
sources and the analyzed metamodel instances is crucial for the further manipulation of the model.

## ECLIPSE

The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. The parsed metamodel instance can be opened in Eclipse. The PL/1 view shows the XML Structure as PL/1 Code Figure 5 below.



Besides that, the Ecore generated graphics of the metamodel instance can be opened in the graphical editor. Now in this editor we have many types of model manipulation functions. Initially the classes correspond to the modules of the program and the links correspond to the program calls. The attributes which are used are shown on each module and can be hidden. Defined functions allow that the modules can be copied together into a new module or content of one module can be divided into two or more modules. Figure 6 depicts the diagram representing the ecore model which shows the initial state of the program. A simple example how codes can be restructured is being shown on figure 7. The module 'handle-state-start' can be eliminated and the code of this module can be migrated into the calling module 'Main'. The graphic is synchronized with the model instance. The next step is the conversion. In this case the pl1 instance is converted to the CA Gen instance. Figure 7 depicts the differences in the generated CA gen instances.



```

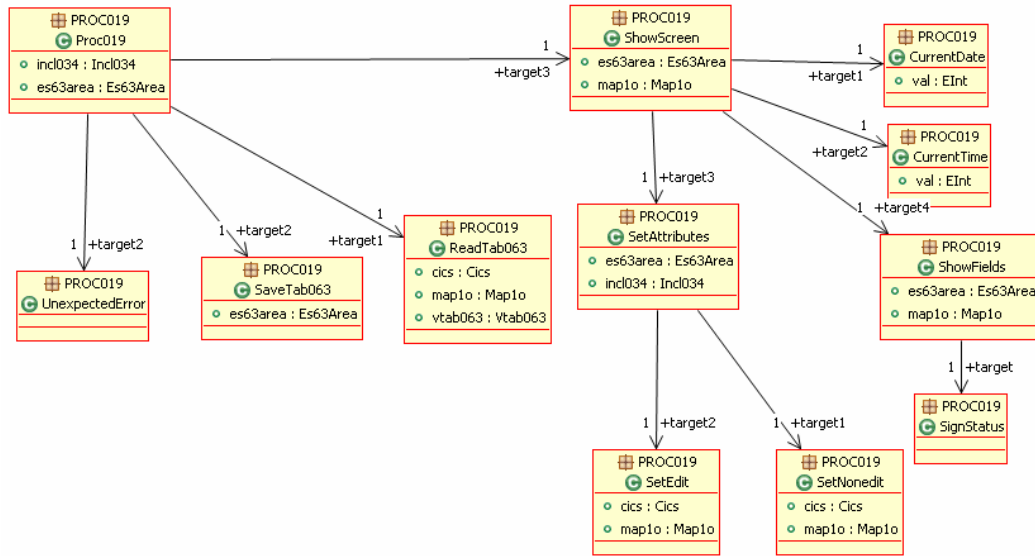
NOTE * -----
* SELECT ( ES63_STATE );
* WHEN ( ES63_STATE_START ) CALL HANDLE_STATE_START;
* OTHERWISE CALL UNEXPECTED_ERROR;
* END;
* -----

IF loc incl034_1 es63_state IS EQUAL TO loc incl034_2 es63_state_start
USE ab_handle_state_start
  WHICH IMPORTS: Work View   loc incl034_2 TO Work View   imp incl034_2
                Work View   loc incl034_1 TO Work View   imp incl034_1
                Entity View  loc vtab063 TO Entity View  imp vtab063
                Work View    loc cics TO Work View    imp cics
                Work View    exp scrn001_2 TO Work View  imp scrn001_2
                Work View    loc proc019_sign_status TO Work View  imp proc019_sign_status
  WHICH EXPORTS: Work View   loc incl034_2 FROM Work View  exp incl034_2
                Work View   loc incl034_1 FROM Work View  exp incl034_1
                Entity View  loc vtab063 FROM Entity View exp vtab063
                Work View    loc cics FROM Work View    exp cics
                Work View    exp scrn001_2 FROM Work View  exp scrn001_2
                Work View    loc proc019_sign_status FROM Work View  exp proc019_sign_status
  IF EXITSTATE IS EQUAL TO link_to_screen
  ESCAPE

```

Figure 6

An example of code restructuring can be exercised with the transformed code into CA Gen. The procedure HandleStateStart has three LOC and can be copied back into the PROC019 Main. The function <inline> can be used to accomplish this task. Note that the procedure calling parameters and variables are being copied automatically with the relocated code to the destination in an EMF editor. In a text editor as we had in the preprocessing stage, one has to take care himself that all related artifacts are being copied.



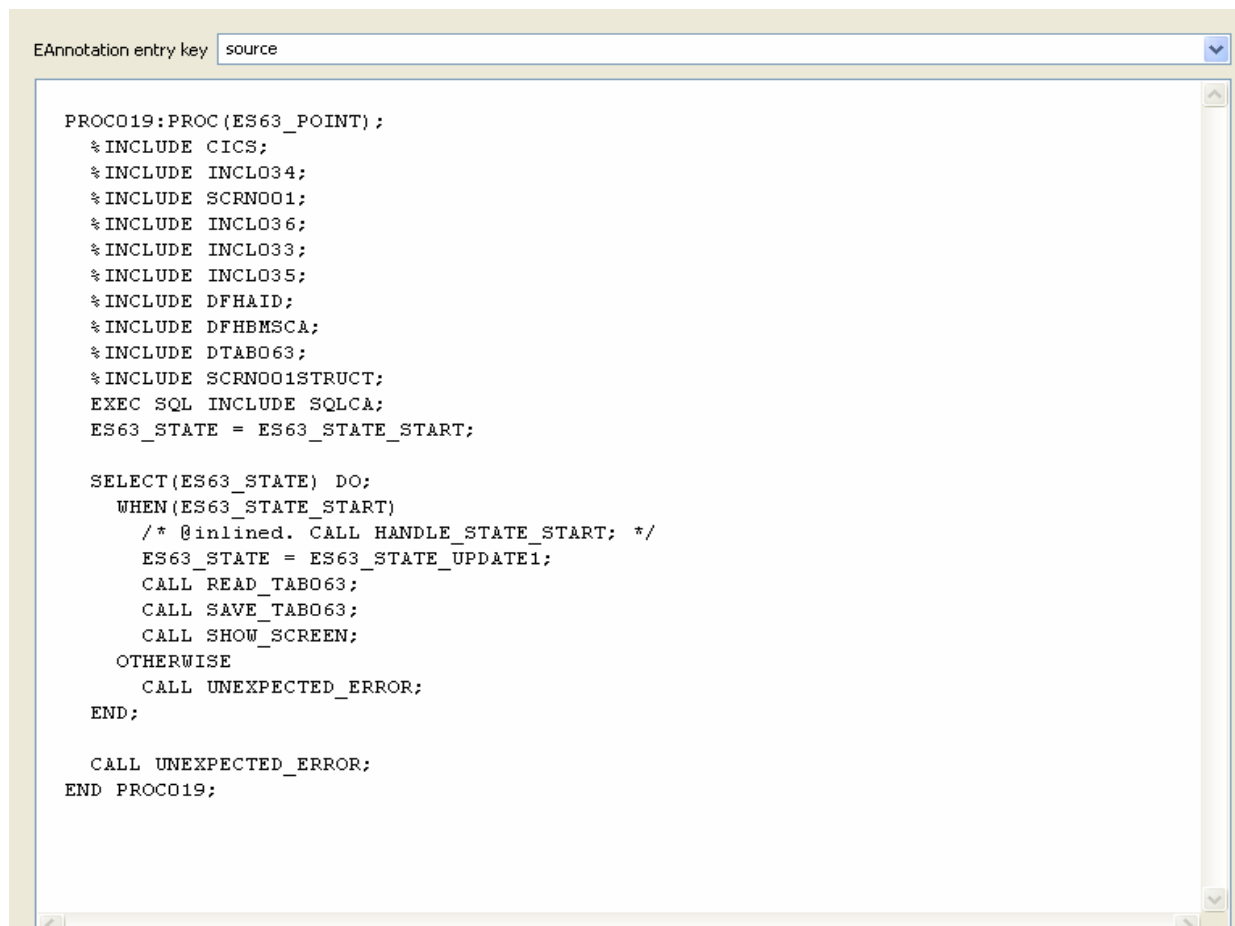
```

* -----
IF loc incl034_1 es63_state IS EQUAL TO loc incl034_2 es63_state_start
SET loc incl034_1 es63_state TO loc incl034_2 es63_state_update1
USE ab_read_tab063
  WHICH IMPORTS: Entity View loc vtab063 TO Entity View imp vtab063
                Work View loc cics TO Work View imp cics
                Work View exp scrn001_2 TO Work View imp scrn001_2
  WHICH EXPORTS: Entity View loc vtab063 FROM Entity View exp vtab063
                Work View loc cics FROM Work View exp cics
                Work View exp scrn001_2 FROM Work View exp scrn001_2
  IF EXITSTATE IS EQUAL TO link_to_screen
  ESCAPE
  USE ab_save_tab063
  WHICH IMPORTS: Entity View loc vtab063 TO Entity View imp vtab063
                Work View loc incl034_1 TO Work View imp incl034_1
  WHICH EXPORTS: Entity View loc vtab063 FROM Entity View exp vtab063
                Work View loc incl034_1 FROM Work View exp incl034_1
  IF EXITSTATE IS EQUAL TO link_to_screen
  ESCAPE
  USE ab_show_screen
  WHICH IMPORTS: Work View loc incl034_1 TO Work View imp incl034_1
                Work View exp scrn001_2 TO Work View imp scrn001_2
                Work View loc incl034_2 TO Work View imp incl034_2
                loc cics TO Work View imp cics
                Work View loc proc019_sign_status TO Work View imp proc019_sign_status
  WHICH EXPORTS: Work View loc incl034_1 FROM Work View exp incl034_1
                Work View exp scrn001_2 FROM Work View exp scrn001_2
                Work View loc incl034_2 FROM Work View exp incl034_2
                Work View loc cics FROM Work View exp cics
                Work View loc proc019_sign_status FROM Work View exp proc019_sign_status
  IF EXITSTATE IS EQUAL TO link_to_screen
  ESCAPE

```

Figure 7

In this picture procedure HandleStateStart is eliminated. The calls to the procedures readTab063, SaveTab063 and ShowScreen are moved to the procedure proc019. The same change is automatically done in the PL1 instance, as you see in the PL/1 instance view in eclipse.(Figure 8)



```
EAnnotation entry key source

PROC019:PROC (ES63_POINT) ;
  %INCLUDE CICS;
  %INCLUDE INCL034;
  %INCLUDE SCRNO01;
  %INCLUDE INCL036;
  %INCLUDE INCL033;
  %INCLUDE INCL035;
  %INCLUDE DFHAID;
  %INCLUDE DFHBMSCA;
  %INCLUDE DTAB063;
  %INCLUDE SCRNO01STRUCT;
  EXEC SQL INCLUDE SQLCA;
  ES63_STATE = ES63_STATE_START;

  SELECT(ES63_STATE) DO;
    WHEN(ES63_STATE_START)
      /* @inlined. CALL HANDLE_STATE_START; */
      ES63_STATE = ES63_STATE_UPDATE1;
      CALL READ_TAB063;
      CALL SAVE_TAB063;
      CALL SHOW_SCREEN;
    OTHERWISE
      CALL UNEXPECTED_ERROR;
  END;

  CALL UNEXPECTED_ERROR;
END PROC019;
```

Figure 8

## Conclusion

In this paper we discussed the Code Restructuring Functionality of a Reverse Engineering Platform. ModelCVS \*) ProgGen is such a system. ProgGen can convert and improve PL/1, COBOL, CAGen and partly JAVA Code (ADABAS NATURAL and RPG are potential extensions) and can manage paradigm change from structural, procedural analysis to object oriented design and analysis. In the following paper we will discuss and explain the usage of the EMF editor to support the paradigm change, OOD and JAVA generation.

APG ModelCVS Team, Vienna 20.4.2009

\*)

This work has been awarded and partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under Grant FIT-IT-810806.