



COBOL REENGINEERING USING THE
PARAMETER BASED OBJECT IDENTIFICATION
(PBOI) METHODOLOGY

THESIS

Sonia de Jesus Rodrigues, Captain, Brazilian Air Force

AFIT/GCS/ENG/99J-02

19990629 051

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/99J-02

COBOL REENGINEERING USING THE
PARAMETER BASED OBJECT IDENTIFICATION
(PBOI) METHODOLOGY

THESIS

Sonia de Jesus Rodrigues, Captain, Brazilian Air Force

AFIT/GCS/ENG/99J-02

Approved for public release; distribution unlimited

DTIC QUALITY INSPECTED 4

The views expressed in this thesis are those of the author and do not reflect the official policy or position of Ministerio da Aeronautica do Brasil.

AFIT/GCS/ENG/99J-02

COBOL REENGINEERING USING THE PARAMETER BASED OBJECT
IDENTIFICATION (PBOI) METHODOLOGY

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems

Sonia de Jesus Rodrigues, B.S.
Captain, Brazilian Air Force

June 1999

Approved for public release; distribution unlimited

COBOL REENGINEERING USING THE PARAMETER BASED OBJECT
IDENTIFICATION (PBOI) METHODOLOGY

Sonia de Jesus Rodrigues, B.S.

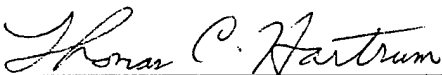
Captain, Brazilian Air Force

Approved:



Dr. Robert P. Graham, Jr., Chairman

2 June 99
date



Dr. Thomas C. Hartum

2 June 99
date



Dr. Scott A. DeLoach

4 June 99
date

Acknowledgements

This research was possible only with the help and support of many. First, I would like to thank my family for the love and support which not only made this research possible, but also worthwhile and my godchildren Clara Rodrigues and Carlos Eduardo Marcello for their letters and drawings that gave me breaks during my long hours of work. I would like to thank my friends Captain Diná Moraes and Major Ferreira Gomes and his family for being so supporting, plus the entire supporting cast from the International Military Student Office, but most specifically Mrs. Annette Robb. Thanks also go to Major Talbert, my academic advisor who always took time to help me relax wherever we met at the Hawkeye lab, and to Dave Doak for being both a source of assistance at lab and a kind person. I would like to thank LtCol Lino Cruz my chief in the Brazilian Air Force(FAB), for being the first person to believe in me, even before I believed in myself and who gave me - with others members of the FAB - the opportunity to do this Master's Degree. I would also like to thank the members of my committee, Dr. Hartrum and Maj. Scott Deloach for making the KBSE meetings worthwhile. I would like thank Captain Diná who worked with me on the translation system phase and shared many weekends and nights with me at the Hawkeye lab as we both strove to understand Cobol Refine software. The results from the work on this phase form part of both our respective theses. I would like to express my deep admiration to the faculty and staff of the Air Force Institute of Technology, School of Engineering for helping me to attain my goals and for their dedication and enthusiasm in teaching and research. Finally I would like to express my appreciation for the three basic supports needed to accomplish this mission at AFIT: my advisor Major Robert Graham, who put his trust in me as a student

to carry out this research, and provided a steady stream of mental guidance. His assistance laid the foundation for all the research represented in this thesis, and I thank him for his unflagging support despite the strains imposed on him. Next, I thank my English teacher Paul Carbonaro for being my sense of vocabulary and expression. Without him, my dissertation would not be what it is today. His effort in, and dedication to understanding my work has been more than a professional. I must thank my dearest friend LtCol William Atella for providing emotional support. He gave me the initial encouragement to come and study in the US. Even many miles away, he has always been “present” to give me continued support and strength. He believed in me and gave me the emotional push needed to accomplish this mission even though I am far from my family, friends and country.

Lastly, I must offer my gratitude to God for giving me strength when I needed it most.

This thesis is dedicated to my late father.

Table of Contents

Acknowledgments	iii
List of Figures	vii
List of Tables	ix
Abstract	x
I. Introduction	1
I.1. Background	1
I.2. Problem Statement	6
I.3. Overview of the rest of the document	8
II. Literature Review	9
III. Methodology	25
III.1. Overview	26
III.2. Approach of the Translation System	26
III.3. Cobol versus GIM Characteristics and Restrictions	29
III.4. Reengineering Methodology	33
III.5. Methodology Conclusion	34
IV. Design of the Reengineering System	35
IV.1. Overview	35
IV.2. Classification of the Cobol Statements	36
IV.3. The Transformation System	36
IV.4. The Translation System	54
IV.5. Modifications to the PBOI Prototype	69
IV.6. Summary	71

V. Analysis of the Methodology applied to FAB Cobol Legacy System	74
V.1. The Brazilian Air Force Cobol Legacy System Transformation	74
V.2. Converting Cobol System to the GIM	74
V.3. Converting GIM to the GOM	75
V.4. Summary	101
VI. Conclusions and Suggestions	103
VI.1. Introduction	103
VI.2. GIM conclusions	103
VI.3. GOM conclusion	105
VI.4. Parameter-Based Object Identification Method Conclusion	106
VI.5. Contributions	108
Appendix A. Cobol Legacy System	110
Appendix B. Legacy System Imperative Code	121
Bibliography	134
Vita	137

List of Figures

Figure	Page
1. Reengineering Process	2
2. Overall View of Reengineering Methodology	5
3. GIM Domain Model	6
4. GOM Domain Model	6
5. Overall View of Research	8
6. PBOI Case Example	11
7. Overall View of Transformation and Translation Systems	35
8. Imperative Data Type Transformation	59
9. Imperative Arithmetic Expression Transformation	60
10. Imperative Conditional Expression Transformation	61
11. Imperative Input/Output Transformation	62
12. Imp-Subprogram-Call Transformation	63
13. System Diagram	77
14. Sigma 3 Conversion Example	91
15. Sigma 3 Conversion Example (CLASS-15)	92
16. Sigma 3 Conversion Example(CLASS-8)	93
17. Sigma 3 Conversion Example(CLASS-17)	94
18. Sigma 3 Conversion Example(CLASS-31)	95
19. Initial Class-System	97
20. Final Class-System	97
21. New Class Originated from Overlapping Classes	98

22. Loss of Functionality – Slicing Problem	100
23. Loss of Functionality – Messages Placed Incorrectly	102
24. Gom-record	106

Table	Page
1. Subprogram Categories	9
2. PBOI Cases	10
3. Cobol Constructs Classification	37
4. Cobol Constructs Recognized by the Translation System	67
5. Cobol Constructs X GIM Constructs	72
6. Sliced Subprograms	78
7. Category Subprograms and Produced Output Parameters	83

Abstract

This research focuses on how to reengineer Cobol legacy systems into object-oriented systems using Sward's Parameter Based Object Identification (PBOI) methodology. The method is based on relating categories of imperative subprograms to classes written in object-oriented language based on how parameters are handled and shared among them. The input language of PBOI is a canonical form called the generic imperative model (GIM), which is an abstract syntax tree (AST) representation of a simple imperative programming language. The output is another AST, the generic object model (GOM), a generic object oriented language. Conventional languages must be translated into the GIM to use PBOI. The first step in this research is to analyze and classify Cobol constructs. The second step is to develop Refine programs to perform the translation of Cobol programs into the GIM. The third step is to use the PBOI prototype system to transform the imperative model in the GIM into the GOM. The final step is to perform a validation of the objects extracted, analyze the system functionally, and evaluate the PBOI methodology in terms of the case study.

COBOL REENGINEERING USING THE PARAMETER BASED OBJECT IDENTIFICATION (PBOI) METHODOLOGY

I. Introduction

1.1 Background.

Organizations have many legacy systems performing crucial work that may represent years of accumulated experience and knowledge. A legacy system is a large software system and might be written in assembly or third-generation language. The systems are becoming too expensive to maintain and simply replacing them may also be too expensive. So, reengineering should support examination and alteration of a legacy system to reconstitute or implement it into a new form [2].

Reengineering is a technique that is becoming more and more important. The interest in reengineering is originated by the need to leverage legacy systems. Previous activities associated with legacy systems were just maintenance with small localized changes until the systems were replaced. Systems were changed to correct bugs or to support new requirements.

Reengineering is the examination and alteration of a subject system to reconstitute it in a new form, followed by the implementation of the new form [2]. Reengineering generally includes some form of reverse engineering (to achieve a more abstract description) followed by some form of forward engineering or restructuring [2]. Reverse engineering can be characterized as analyzing software to identify the system components and their interactions, and represent the system on a high level of abstraction.

Figure 1 shows a generalized view of the process of reengineering legacy code as developed by Byrne [1].

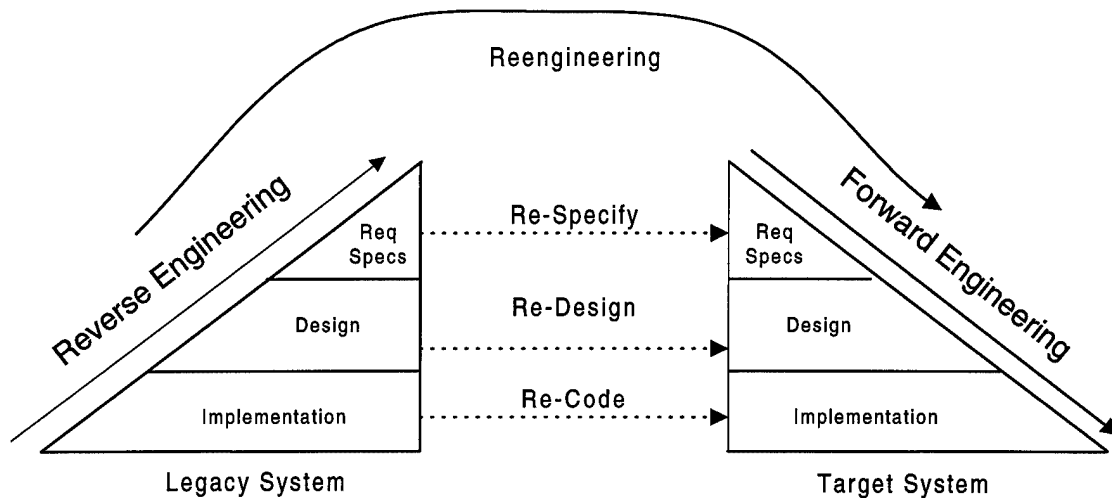


Figure 1 Reengineering Process

Nowadays, legacy systems that are in use in several military units and other business organizations play fundamental parts and have great credibility for the users. Most of the existing systems are mainframe and Cobol-based. Some of the common problems presented by those systems include unstructured code, inefficient execution, difficulty of maintenance, bad documentation and complexity. Those problems cause great damage to the businesses. Therefore, the systems should be migrated by using a paradigm that makes better performance, easy maintenance and reusability possible.

The object-oriented paradigm with its promise of re-usability, extensibility, and maintainability has great appeal to organizations and encourages them to exchange their

legacy systems. Korson and McGregor [5] characterize the object-oriented paradigm using the following concepts:

Classes - A class is a template that defines the attributes and operations for each instance of the class.

Objects - Object is an instance of a class. Objects model real-world entities that have state, behavior, and identity.

Methods - A method is a sequence of object-oriented statements that implement a specific behavior.

Messages - A message invokes a specific method in an object. Messages are sent to a target object that must be able to execute the method being invoked.

Inheritance - The classes in an object-oriented design are organized in a class hierarchy where certain classes inherit the attributes and operations from other classes in the hierarchy.

Polymorphism - In an object-oriented design, it is possible to have methods (from different classes) with the same name. Polymorphism means the appropriate method will be executed based on the class of an object instance.

Typical legacy systems are written in some imperative program language, such as Fortran or Cobol. System maintenance is done and its documentation and structure degraded, so the only reliable source of information about it is the source code. Therefore, the reengineering must involve reverse engineering to increase understanding in design level and create representations for it. After reverse engineering, forward engineering should be applied for renovation of the programs into an object-oriented language.

Reverse engineering must apply some techniques to determine the abstract elements and extract objects. There are several techniques for understanding program constructs and identifying objects. One is the Global Based Object Identification (GBOI) technique, which establishes links to routines that manipulate global and static data [3]. Another one, Type Based Object Identification (TBOI), establishes relationships between data types and routines that use them for formal parameter or return values [3]. The Parameter Based Object Identification (PBOI) was defined by Major Sward in his thesis "Extracting Functionally Equivalent Object-Oriented Designs from Legacy Imperative Code" [19]. It is based on relating categories of imperative subprograms into classes, based on how parameters are handled and shared among them. The PBOI method provides a rationale for converting imperative subprograms into classes and methods that implement the subprograms. Figure 2 shows the overall view of this methodology [6]. PBOI was developed with Fortran in mind, since Fortran for most of its history and usage lacks the elaborate type definition capabilities that Cobol and other imperative languages have and on which techniques such as TBOI depend. Despite this mindset, PBOI was designed to be applicable to any imperative program.

The input language of PBOI is a canonical form called the generic imperative model (GIM), which is an abstract syntax tree (AST) representation of a simple imperative programming language. The GIM models the variables, expressions, assignment statements and control flow typically built into imperative programming language. Figure 3 shows a partial representation of the GIM domain model. Conventional languages must be translated into the GIM to use PBOI. Sward demonstrated this by writing a Fortran to GIM translator. The output is another AST, the

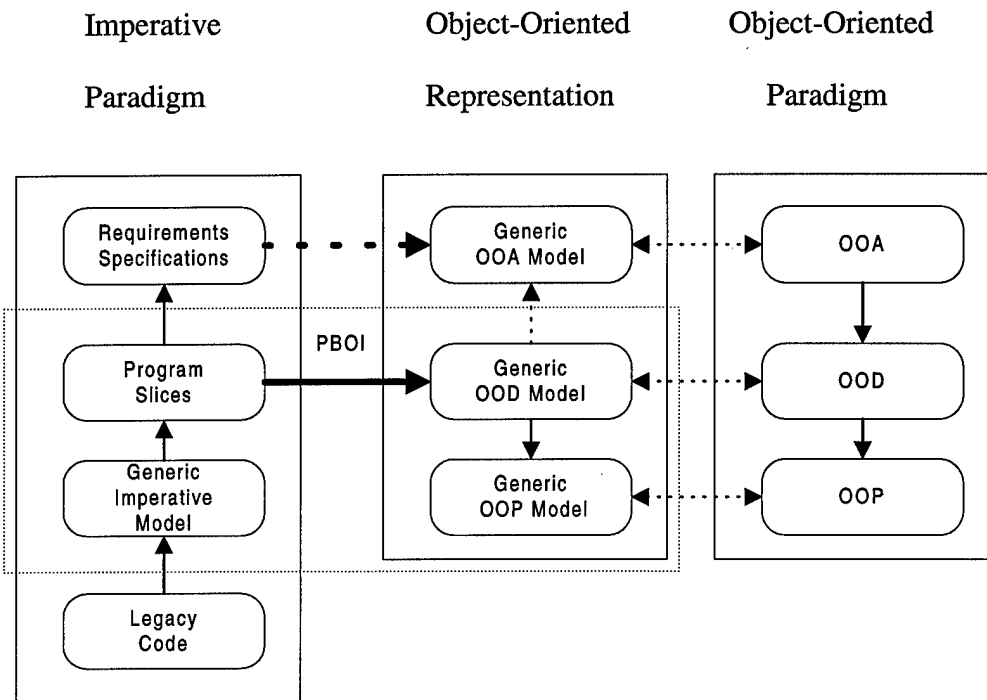


Figure 2 Overall View of Reengineering Methodology

generic object model (GOM); a canonical generic object oriented language. The GOM models objects, classes, methods and messages typically built into an object-oriented programming language. Figure 4 shows a partial representation of the GOM domain model. The GOM must be translated into a conventional language, such as ADA, C++ or Java, for compilation and execution.

Sward's claim is that many languages, such as Ada, C, Pascal or Cobol could also be translated and PBOI applied. My research objective is to determine whether or not PBOI is a viable tool for reverse engineering Cobol systems.

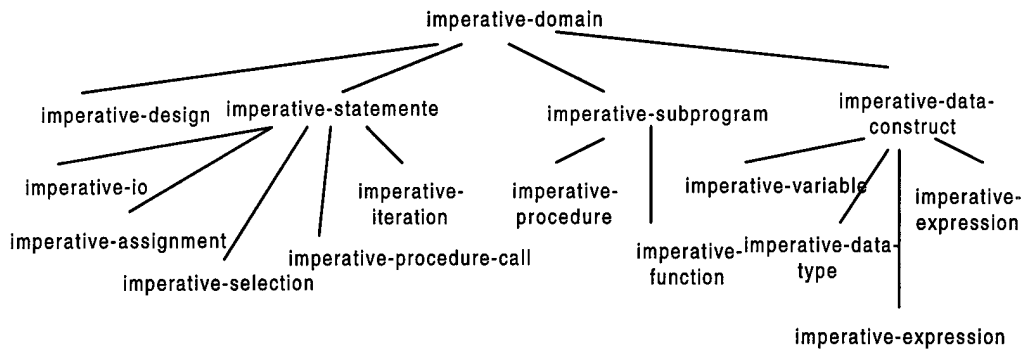


Figure 3 GIM Domain Model

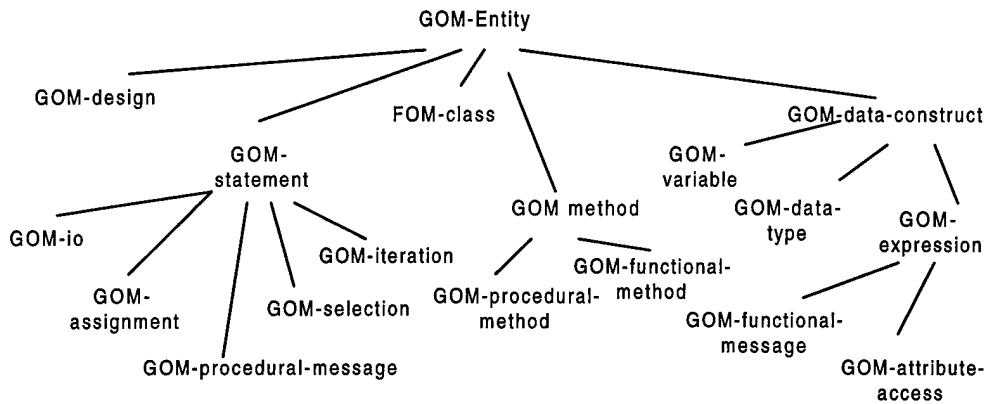


Figure 4 GOM Domain Model

1.2 Problem Statement.

This research focuses on how to perform reengineering of Cobol legacy systems into object-oriented systems using the PBOI methodology. PBOI formal transformations

extract an object-oriented design equivalent to the legacy imperative code and it is feasible to automate this methodology. The Sward dissertation was based on legacy Fortran imperative code. The objective of the research is to evaluate the methodology that Sward developed, to determine whether or not it is a viable tool for reverse engineering Cobol systems.

The GIM is programming language independent; in this way, the GIM allows the PBOI prototype to be easily extended to other languages. The first step of the research is to translate Cobol code into the Generic Imperative Model (GIM) Abstract Syntax Tree (AST). So, it is necessary to construct an automatic transformation system. The translation part of the thesis was done in collaboration with Captain Diná Moraes (FAB). Her research then evaluated the ability of the GIM to handle the Cobol language and proposed some changes [24].

The second step is to extract an object-oriented design by using the PBOI methodology, as currently implemented by Sward. The extracted object code is represented in the GOM, which has been developed to model objects, classes, methods and messages.

The third step is to analyze the extracted objects and verify their consistency with the original imperative code to validate that the object oriented design is functionally equivalent to the legacy system, as Sward claims he has proven.

The fourth step is to analyze the objects to see if they constitute a reasonable or plausible object-oriented design, or at least can serve as a starting point for further design refinement.

Figure 5 shows an overall view of this research.

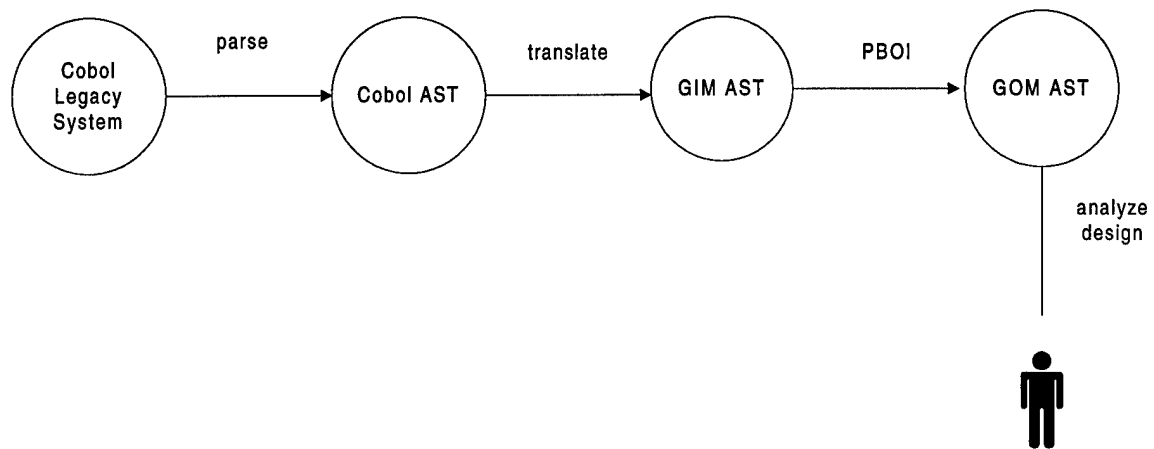


Figure 5 Overall View of Research

1.3 Overview of the rest of the document.

The remainder of this thesis proceeds as follows. Chapter II reviews previous work in the area of reengineering. Chapter III describes the methodology used to transform a Cobol legacy system into the GOM. Chapter IV presents the design of the transformation and translation systems with the classification of the Cobol constructs, and also describes the PBOI prototype. Chapter V describes the Brazilian Air Force Cobol legacy system transformation into the GOM. Chapter VI presents conclusions about GIM, GOM and PBOI methodology.

II. Literature Review

This section reviews previous work in the area of reengineering. This review includes approaches in reverse and forward engineering. Reverse engineering supports reengineering, and forward engineering supports the implementation of a new system with the same functionality as the legacy system.

2.1 Sward's work is based on PBOI methodology [19]. The PBOI methodology classifies all imperative subprograms into six categories. Table 1 shows this classification.

Table 1 Subprogram Categories

Number of Calls to other Subprograms	Zero	Greater than zero
Number of Data Items produced by the Subprogram		
Zero	Category 0	Category 1
One	Category 2	Category 3
Greater than one	Category 4	Category 5

The processes of slicing and masking convert the category 4 and 5 subprograms into category 2 and category 3 subprograms. The slicing process builds one program for each output parameter, and each program is composed of the statements involved in changing the value of the data item produced in that subprogram. The masking process creates local variables. They substitute the variables that are different from the one

produced in the subprogram, and which are involved in the slicing that transforms the subprogram into category 2 or 3.

After, the procedures are converted into methods and classes.

For subprograms in category 2, the formal parameters are converted into attributes of a class and the subprogram is converted into a method of the class.

For category 3 subprograms, the subprogram is converted into a method of the class and initially the formal parameters are converted into attributes of a class. Later, the attribute can be converted into parameters of the calling method or of the called methods. The filtering to determine which parameter will be converted into an attribute (or a parameter of another class) is based on the classification of the parameters. The PBOI methodology classifies the subprogram parameters into four cases. Table 2 shows this classification.

Table 2 PBOI CASES (parameter classification)

	Actual in the called subp. is Formal in the calling subp.	Actual in the called subp. is not Formal in the calling subp
Formal in the called subprog. Is Attribute in the called subprogram/class	PBOI CASE 1	PBOI CASE 3
Formal in the called subprog. Is Parameter in the called subprogram/class	PBOI CASE 2	PBOI CASE 4

Consider the example below of two imperative subprograms (Figure 6) and the class that was converted from the subprogram. The subprogram PGM-0220 is a category 2 subprogram, so the formal parameters are converted into attributes of a class and the subprogram is converted into a method of the class. Next, to convert the subprogram PGM-0210-400036-AV-400010-TABLE to a method and class, the parameters of the calling and called subprograms are classified to determine how to convert the two subprograms.

```
Procedure PGM-0210-400036-AV-400010-TABLE( 400780-INDEX, HEX-1,  
      400033-LOC-400010-TABLE, 400036-AV-400010-TABLE)  
begin  
  LOCAL-1 := 400780-INDEX;  
  if 400033-LOC-400010-TABLE ( 1) = "VASP"  
  then LOCAL-1 := HEX-1;  
      PGM-0220 ( 400036-AV-400010-TABLE, LOCAL-1)  
  Else endif  
End
```

```
Procedure PGM-0220 ( 400036-AV-400010-TABLE, 400780-INDEX)  
Begin  
  if 400036-AV-400010-TABLE ( 400780-INDEX) = "S.TEC"  
  then 400036-AV-400010-TABLE ( 400780-INDEX) := "VASPT"  
  else 400036-AV-400010-TABLE ( 400780-INDEX) := "VASP " endif  
end
```

Figure 6 PBOI Case Example

The parameter 400036-AV-400010-TABLE is classified as PBOI CASE 1 and LOCAL-1 is classified as PBOI CASE 3. Additionally, C-4 is an instance of CLASS-1 class. Therefore, the final classes and methods converted from the two subprograms are :


```

class CLASS-4 attributes
  400033-LOC-400010-TABLE, HEX-1,
  400780-INDEX
method PGM-0210-400036-AV-400010-TABLE ( C-4,C-5 ) begin
  LOCAL-1 := GET-400780-INDEX ( C-4);
  if GET-400033-LOC-400010-TABLE ( C-4, 1) = "VASP"
  then LOCAL-1 := GET-HEX-1 ( C-4);
    PGM-0220 ( GET-400036-AV-400010-TABLE(C-5), LOCAL-1)
  else endif
end
superclass USER-OBJECT

```

```

class CLASS-1 attributes
  400036-AV-400010-TABLE
method PGM-0220 ( C-1, 400780-INDEX ) begin
  if GET-400036-AV-400010-TABLE
    ( C-1, 400780-INDEX)
    = "S.TEC"
  then SET-400036-AV-400010-TABLE
    ( C-1, 400780-INDEX, "VASPT")
  else
    SET-400036-AV-400010-TABLE
    ( C-1, 400780-INDEX, "VASP ")
  endif
end
superclass USER-OBJECT

```

2.2 Yang's Work. The method reverse engineers Cobol programs into a reusable form through program transformation based on a wide spectrum language called the Reengineering Wide Spectrum Language (RWSL). They use the Reengineering Assistant (RA) prototype to support transformation and semantic interface analysis for reuse of Cobol programs [6].

The method consists of the following steps:

- 1- Translating a Cobol program into RWSL by Translator (an RA tool component).
- 2- Looking for functionally self-contained modules. A reusable component can be obtained from a self-contained module. A self-contained module can be a code module, a function or a procedure in the system.
- 3- Taking each self-contained module and applying program transformations to abstract the module into its high-level representation using Entity Relationship (ER) diagrams.
- 4- Using the ER diagrams together with the original code, use a semantic interface analysis tool to generate semantic predicates and interface predicates for a reusable module in terms of its pre-conditions, post-conditions and obligations.
- 5- Storing the reusable module and maintaining a link between the ER representation and the reusable module.

The method obtains reusable Cobol code components and their designs, written in RWSL, by combining an analysis of data structures and code. It makes the original program more understandable because it represents the abstracted ER diagram. The components saved can be reused but it is necessary that future research in RA applies the reusable components.

In comparison, the PBOI approach is based on obtaining an object-oriented design for the original Cobol system while the aim of Yang's research is a reusable library of components and design.

2.3 Yoshino's method generates a narrative specification used by real-world maintainers to facilitate the understanding of business procedures in existing Cobol programs [7].

This research determines which information should be extracted from Cobol programs for software maintenance. This information is needed to:

- 1- Distinguish normal and error processes, which coexist in systems.
- 2- Assign data items to conditional branches. Convert control-centered expressions in a program into data-centered expressions in the specification.
- 3- Call external subprograms to understand the parameter assignment, invocation and the return code check
- 4- Eliminate temporary variables, and remove statements with temporary variables to make the program description more comprehensible.
- 5- Replace Perform statements by the performed target code when the following restrictions are satisfied: number of statements in the performed code is under a fixed number (100) and the number of the calls of the performed code is below a fixed number (3). Relocate the subroutine to the position where it should have been originally to make the program easier to read.
- 6- Extract numerical and actual specification headings for quick reference.
- 7- Relate branch conditions and their procedures to build a table for the specification.
- 8- Add cross-references to the specification when the process that follows is not on the next line.

2.4 REDO Sneed's work is the result of research conducted at Oxford University on how to transform Cobol programs into object-oriented specifications [8]. The input of this process is a Cobol program without database accesses or special data communication

interfaces. The output is a formal specification in the language Z++. The process is accomplished in three steps. The first is to translate the Cobol program into the UNIFORM language. UNIFORM is a meta-language that facilitates the production of documentation such as data flow, entity-relationship (ER) and others. During the second step, every record type is recognized as an object and every field as an object attribute. The procedure division is divided into slices based on data flow analysis. I/O operations on a particular file and the statements that manipulate the contents of this file are identified and determine a program phase. Phases correspond to data flow paths. The last step generates an object-oriented specification. The program slices produced during the second step are attached to the objects to which they are related, and will become methods in a class. The statements that access, alter or set attributes to records, which belong to a class, are components of the class methods. Finally, the UNIFORM syntax is converted to a Z++ notation. The result of this process is a class specification for each file and the procedurally structured statements are related to the classes.

PBOI research and Sneed's research both have Cobol reengineering as an objective. The aim of both sets of research is to reconstruct the Cobol system in an Object-Oriented model. These sets of research are based on two phases. One is to transform the program into an intermediate structure: GIM for PBOI, and UNIFORM for Sneed's. GIM and UNIFORM can be seen as canonical languages. Sneed's research uses the UNIFORM to produce technical documents, and PBOI methodology uses GIM to translate the system into the GOM. In Sneed's research, the records are used to identify objects, of which every field becomes an attribute, and slices are cut up from the Procedure Division. The slices are a sequence of statements from the file input to the file

output. Later, the slices are attached to the objects to which they refer. So, Sneed's method is based on record identification, while the PBOI methodology is based on parameter identification. The GIM lacks record types, so this information is unavailable to PBOI.

Sneed's method is similar to the TBOI method, because both identify the classes based on the types of formal parameters and the operations that manipulate them [3].

2.5 Fantechi's work relies on using a tool (C_2O^2) for analyzing Cobol applications [9]. A software prototype was developed based on a Lex/Yacc engine, which is capable of processing all Cobol syntax and semantics. The software prototype was implemented using the following method. Single Cobol programs are classified as subprograms, batch programs and online programs. Main programs can be batch and online programs. The basic idea in this approach to extracting object-oriented analysis from a Cobol application is to focus on the Data Division that contains the information to create a representation of the data structures. The entire transformation process, from Cobol application to an object-oriented design, is realized in five phases. In the first transformation of the main program identifies the corresponding classes. This process begins by an analysis of all the data structures of the application's modules by identifying the minimal number of data structures that are considered early prototypes of classes. The minimal number of data structures is identified by eliminating the redundant definition of those structures. The elimination is based on synonyms, numeric suffixes or another convention used in the Cobol program.

The second phase establishes relationships of aggregation, association and specialization among early prototype classes by which to organize them into classes. The third phase of the transformation process is based on the analysis of the accesses to data, to determine the relationships between classes and to assign access methods to the class members. In the fourth and fifth phases, the code is reallocated to classes and methods are organized. The first three phases involve the reanalysis of the system.

2.6 The objective of Boyle's research is to focus on Cobol reengineering, specifically the restructuring of Cobol programs [22]. For this restructuring, the author built a system based on transformations and derivations. These transformations and derivations are based on knowledge about a particular Cobol programming style, program environment, or good programming practice.

The methodology described by Boyle transforms the Cobol program into an intermediate language, making it unambiguous, more self-documenting and easier to understand the control flow. The restructuring of the program in that intermediate language is accomplished with the objective of making the program modular and top-down structured. That restructuring uses the transformation technique of unfolding and folding. Paragraphs called by perform statements are transformed in procedures, while paragraphs that are called by GO TO statements continue being paragraphs. In other words, all the implemented transformations are based on a certain knowledge criterion that makes the program most easily restructured. The last phase of that methodology is to generate a structured Cobol program, using the program stored in that intermediate

language. The system that accomplishes that reengineering is based on transformations and derivations and was implemented in TAMPR.

Boyle's research is composed of two different phases.

1. The first phase is the transformation phase that is responsible for including more understanding of the behavior of the program and improving the readability and understandability. So, the program is restructured. Subsequently, this Cobol program is converted into a simplified language.
2. The second phase is the transformation of the program written in simple language, for Cobol language again. The system is implemented using TAMPR and based on transformations. The final product is a new structured Cobol program.

The TAMPR transformations seek a pattern that comprises the structures/statements of the language in which the program is written. When the TAMPR finds the pattern, it changes it by another structure defined by the engineer. Both sets of software can apply transformation sequences.

The author uses canonical forms to build different constructs in only one way. That way represents several statements and facilitates the final transformation of the program and the generation of that program into a specific reengineering aim. The canonical forms are also used to structure the program. Some canonical forms are structured into conditional statements and loops.

Reading Boyle's paper, it is clear that he intends to develop a tool capable of improving the structure of Cobol programs. From my point of view, the research almost has complete success, since the generated final program is easier to understand and more

modular than the original. However, I don't agree with the author that the program is completely structured because, in the final program, there is a loop structure that has different exits. So, it is possible to exit the loop structure not just by the loop condition test. That is, in my opinion, a flaw in structured programming.

Like PBOI and Sneed, Boyle's work is based on two distinguished phases. The first phase is to transform the program into an intermediate structure. The second is to implement the reengineering. The intermediate structure is a canonical form analogous to the GIM. This intermediate structure, then, can be used to reconstruct a new program. In other words, it does not matter which the original language of the program is. After the Cobol program goes into the intermediate structure, it is possible to reengineer it. In the case of Boyle's 1998 research, the reengineering is for the same Cobol language. In contrast, my research is about extracting objects. The research effort makes the program easily understood, by renaming Cobol structures, and eliminating or duplicating code to turn the program into modulate and top-down structure. In the PBOI research, the translation of the legacy system into the GIM does not take into consideration the best understanding or structure of the programs, except that the object-oriented form will be better somehow. The two research efforts use systems based on transformations. The research for restructuring Cobol programs concludes the reengineering and generates a source program in a programming language (Cobol). In contrast, the PBOI methodology does not generate a new program using any language. Boyle's approach is based on a particular Cobol programming style while Sneed's is based on recognizing a record type as an object. Then again, PBOI is more generic than both approaches, because it does not take into consideration a specific programming style or a specific data type.

2.7 Livadas's research specifies a new approach to finding objects in programs [10]. They introduce the idea of two-step object identification and the idea of receiver-based object identification. The aim of secondary object finding methods is to construct secondary object groupings from those produced by RBOI. The receiver-based object identification (RBOI) extracts candidate objects based on a receiver parameter type. A receiver parameter type is one which is modified inside a routine.

The RBOI clusters a routine with the types of its receivers. The RBOI can be applied to global and static variables. A candidate object in a program P relative to a method M is defined as a triple $C_m^P = (\phi, \mathfrak{R}, \delta)$ where ϕ is a subset of routines, \mathfrak{R} is a subset of receiver types and δ is a subset of data items. In the secondary object finding methods, there are some operations such as: selection, union, intersection, subtraction and deletion. The method is similar to relational data base queries and the queries help to refine the object groupings. With a large set of types produced by RBOI or other primary identification, this query can cluster the routines with the most complex types. The complexity relation forms a directed acyclic graph on the set of types. The first step in the method is to model a grammar to construct the internal program representation; that is, the system dependence graph (SDG). The SDG models a grammar that permits primitive data types, records, while, for loops, goto continue and break statements. Yet, the SDG does not support pointer variables. The methodology in this research is similar to that of PBOI because it is based on subprogram parameters.

2.8 De Lucia's research proposes a method for migrating legacy systems into an object-oriented platform. The approach is based on the Encapsulation, Reengineering and Coexistence of Object with Legacy (ERCOLE) project of the University of Salerno [11]. This project provides strategy and supporting technology to migrate legacy systems toward object-oriented platforms. Most tools supporting the ERCOLE have already been implemented, but some are still in progress. The process of migration has six steps and is based on reverse engineering and reengineering. The reverse engineering phase decomposes the programs into components that implement user interface management, and those that implement application domain objects. The reengineering phase activities use wrapping techniques. These techniques facilitate the new system by using existing resources, and they allow identification and translation of the objects to be carried out incrementally. So, a new object-oriented system and a legacy system coexist. The objects are identified and encapsulated into an object wrapper. Thus, the new system can use the existing resources through the interface's wrapper. The last step is an incremental translation of the object wrappers, identified in the previous steps, using an object-oriented language. The first step, Static Analysis of Legacy Code, is responsible for extracting all the information needed for the next steps. The information is recovered by several static analyzers, which cover different versions of RPG/400 and embedded SQL code. The analyzers were implemented using YACC facilities and the Visual Age C++ for the OS/2 environment. Information about the system such as control flow graph, variables and where they are used, the embedded SQL code treated as a single node of the system RPG and the related SQL section information, program calls, record structures, files, arrays, key, and parameter list are stored in DB2 tables. The second

step, Decomposing Non-Batch Programs, is responsible for decomposing iterative programs in interface management, components and application domain components. This decomposition allows the system to be reengineered in a client-server paradigm. A tool to build a control dependence graph and a slicer supports the process in this step. The slicer analyzes control dependencies and calls among subroutines to identify the statements involved in implementing the interface manager component. The statements that implement rules and contain data base accesses are identified as application domain components. In the third step, Abstracting an Object-Oriented Mode, batch programs and the application domain components, extracted in the second step, are analyzed to determine an object-oriented model. The approach for identifying the state of the object is based on persistent data stores, and identifying object method candidates is based on chunks of the code. After identifying the data stores that determine the object state, programs, subprograms (or set of), and slices are analyzed to assign them to object methods. The coupling measurement is based on the computation of the accesses of program to data stores. The associations are achieved based on minimization of the coupling measure. When a program does not access other objects (exclusive coupling), the program is assigned to an object. In this situation, the program is considered a method of the object to which the program has access. When the coupling measure between the program and the object is predominant in respect to the coupling measures between the program and the other objects, the program is assigned to the object. In this situation, the program is considered as a message to the other objects. When the coupling measure of a program is uniformly distributed, the program is analyzed to identify subroutines (or set of) to be candidates for object methods. The analysis is performed to

transform the subroutine graph, constructed during the Decomposing Non-Batch Programs step, into a dominance tree [18]. The coupling measure between subroutines and persistent data stores are computed. The subtrees that contain one or more subroutines (whose coupling measure is exclusive or predominant to the same object), are candidate object methods. It is possible that after analyzing the subroutines, one with a uniform distribution coupling measure can still exist. Thus, slicing techniques [2] are applied to determine chunks of the subroutine to implement methods of different objects. In the fourth step, Reengineering the System According to the Abstraction Results, each subroutine, set of subroutines, or slice is encapsulated into a different program. The identification of the interfaces of these new programs and the reengineering of the database access require special attention. A data flow analyzer and a tool to support software reengineering are implemented to reengineer RPG programs. In the fifth step, Encapsulating Identified Objects within Object wrappers, groups of programs and persistent data store, which implement an object, are encapsulated into an object wrapper. Wrapper interface is a method for each program that implements an object method in the object wrapper. The wrapper interface includes simple get/put operations to access the persistent data stores encapsulated within the object wrapper. Messages received by the object wrapper are converted into a call to a program that implements the function. The calls between programs and access to persistent data stores encapsulated into different object wrappers are not exchanged by messages, because the objects are not in an object-oriented platform. The sixth step, Incremental Translation of Object Wrappers, is still being studied. A tool to support the software engineer in the creation of the C++ is being implemented.

2.9 Leite's work describes an automated transformation from Cobol to C/C++ and shows how to handle transformation in a structured semi-automated manner [23]. This approach is based on the transformational engine DRACO-PUC in porting Cobol programs. DRACO-PUC is a software engine being developed at PUC-Rio (Pontifical Catholic University of Rio de Janeiro), that uses the ideas of the DRACO paradigm [23]. DRACO-PUC is based on a powerful transformation engine that is the basis for the transformation strategy. The DRACO-PUC transformations allow local transformations that are applied to short segments of a program and global transformations that are applied to large, distant but related, program blocks. The DRACO-PUC has a parser generator that parses a program into DRACO abstract syntax trees (DASTs). The transformations are performed using the internal representation of DASTs. The first step of the transformation is to parse and generate the DASTs. Second, the transformations are achieved by rule and recognition pattern. The transformations can map descriptions in one language into the same language or into other languages. To accomplish the transformation of the Cobol legacy system, the system is first restructured. Then, the set of paragraphs is grouped in procedures. Analyzing a call graph among procedures helps this activity. The data flow analysis is used to determine which modules will have separate compilations. Next, the conversion of the Cobol program into C/C++ is performed in three more steps. First, the program is divided into blocks according to the control flow analysis. Second, the data division is analyzed and the semantic mapping between the structured Cobol program and C++ is defined. Third, the C++ program generated in the second step is converted into a more readable C++ program.

2.10 Summary.

The approaches to software evolution are changing rapidly along with changing technology. Several approaches have been presented in this chapter that extract objects from legacy systems. Some of them extract specifications to facilitate program understanding.

III. Methodology

3.1 Overview.

This chapter describes the methodology used to transform a Cobol legacy system into the GOM. The methodology presented provides a technical approach for the Cobol reengineering process. The methodology provides a way of extracting programming constructs represented as an AST from Legacy Cobol code, and populating the GIM and GOM. Therefore, the methodology provides a framework for Cobol reengineering, and makes the transformation of a Cobol legacy system into the object-oriented paradigm possible.

3.2 Approach to the Translation System.

A major part of this research is the translation of Cobol code into the GIM AST. The transformation is developed using the Software RefineryTM development environment and the Refine/CobolTM reverse engineering tool.

The translation of Cobol code into the GIM AST is done in two steps: transformation and translation.

The first step of translation is classifying the Cobol constructs into four classes: transformable, directly translatable, indirectly translatable or not handled.

The transformable constructs are not represented in the GIM, but can be rewritten into equivalent Cobol constructs that are directly or indirectly translatable. The transformations will be implemented by developing programs in Refine.

The following statement illustrates an example of a transformable Cobol construct rewritten into an equivalent directly translatable Cobol Construct.

COMPUTE a b = c + d.

This statement computes the sum $c + d$ and places the result in both a and b . The GIM lacks this "multiple assignment" capability. Transforming this statement to

COMPUTE a = c + d.

MOVE a TO b

makes the eventual translation more straightforward.

The following Cobol *PERFORM* statement illustrates an example of a transformable Cobol construct rewritten into an equivalent indirectly translatable Cobol construct.

PERFORM paragraph1 thru end-paragraph1 7 TIMES.

This statement executes the statements that are written within all the paragraphs between *paragraph1* to *end-paragraph1* a total of seven times. Transforming this statement to

PERFORM paragraph1 thru end-paragraph1 VARYING var1 from 1 by 1 UNTIL var1 = 7.

makes the eventual translation more straightforward.

The directly translatable constructs will be converted directly into the GIM, because they are modeled by GIM. These constructs correspond closely to GIM constructs. For example, the Cobol statement

ADD a TO b GIVING c.

corresponds directly to the GIM statement.

$c := a + b$

The indirectly translatable Cobol constructs are not represented in the GIM and have no equivalent Cobol construct that is directly translatable into the GIM. To convert these constructs into the GIM, we have to identify the closest imperative statements to them, and implement this conversion by programming. The following Cobol *PERFORM* statement, used as iteration construct, illustrates an example of an indirectly translatable Cobol construct.

Indirectly-Translatable Cobol Construct:

```
PERFORM sum-of-odd-numbers  
      VARYING temp FROM 1 BY 2  
      UNTIL temp IS > maxodd
```

Imperative Construct:

```
Temp := 1  
  
WHILE temp <= maxodd DO  
  BEGIN  
    sum-of-odd-numbers  
    Temp := temp + 2  
  END
```

The not-handled constructs are not recognized by the GIM and it is difficult or impossible to convert them into constructs that the GIM recognizes.

The Cobol *GOTO* statement illustrates an example of a Cobol construct, that is not handled because the GIM has no *GOTO* statement. Constructs that are not handled by the translator impose restriction on its input: Cobol programs to be translated must first be restructured to remove any occurrence of these constructs.

3.3 Cobol versus GIM Characteristics and Restrictions.

A Cobol program is composed of Divisions, Sections, Paragraphs and Sentences. The translator uses the Identification Division, Data Division and Procedure Division for the transformation of Cobol programs into the GIM. The Environment Division is not used, because this division presents those aspects of the program that depend on the particular hardware to be used and such information is not modeled in the GIM.

The Identification Division is used in the transformations just for the identification of the main program, recovered from the *program-id* paragraph. The GIM does not model documentation nor does it model comments.

The Data Division contains descriptions of the data used by the program, the hierarchical relationships among data, and condition-names. Therefore, all data used inside paragraphs are global variables and can be referenced. The GIM has only local data, so the data items to be used in a procedure (performed paragraphs) must be passed to it as parameters. This division and the Procedure Division are of great importance in the transformation of the Cobol program into the GIM.

The Procedure Division contains the procedures associated with a program. In this division, all statements to be transformed into the GIM and the main program are

identified. Paragraphs that are executed by a *perform* statement are transformed into an imperative subprogram.

The main program is delimited by the *Stop Run* statement. Even though a Cobol program can have more than one *Stop Run* statement, the legacy system must be restructured as outlined in the PBOI methodology. Therefore, the main program is composed of all the statements from the beginning of Procedure Division to *Stop Run*. The *program_id* paragraph identifies the imperative program name.

The imperative subprograms are identified by the existence of *perform* statements. All statements composed between the paragraph name and the thru paragraph name are used to build an imperative subprogram. Therefore, paragraphs found before the *Stop Run* statement and that are executed by *perform* statement continue existing in the imperative main program and a subprogram is created with the corresponding statements. The paragraph name is used to identify the imperative subprogram.

The transform system implementation is restricted to the transformation of a Cobol program with just the initial section. With more than the initial section, the Cobol AST becomes a complex structure. Additionally the information is spread in different tree attributes. So, to retrieve it from the complex structure, and translate the Cobol constructs into the GIM would only serve to increase the complexity of the transformation and translation system. Therefore, the Procedure Division of the Cobol program to be transformed into the GIM cannot be subdivided into sections.

Cobol allows the programmer to build collections of heterogeneous data items. In the File Section and Working-Storage Section of the Data Division, a description with an entry level that is subdivided into other group items or elementary items constructs a

heterogeneous data item. This record structure is an important concept in Cobol. Records are used as operands in several Cobol constructs. Therefore, it is not viable to restrict a legacy Cobol code to not have heterogeneous data, because a Cobol program is heavily based on record structures.

The solution is that the transformation system must implement a transformation to change the records in the Data Division into elementary items. Also, the transformation system implements the alterations to transform the statements that use group items into set of statements that use only elementary items.

For this research, the input Cobol program has to adhere to certain restrictions. Some restriction examples are shown below and all the restrictions imposed on the statements by the difficulty of the transformation are presented in Chapter IV.

One restriction is not to use the *Go To* statement, since the GIM doesn't implement it. Another restriction is not to use *move* statements from group items to group items where the structures are different, or in the condition clause of the *if* and *perform* statements. Consequently they were not implemented into the AST structure. In spite of the fact that the most-used Cobol statements are transformed into the GIM, certain statements have to have their characteristics restricted because of the difference between the GIM AST structure and the Cobol AST structure.

Restrictions on the legacy Cobol program imposed by the GIM are listed below, and explanations about them are provided in Sward's dissertation [19].

- A formal parameter of a procedure must not be both an input and an output parameter. This restriction is not satisfied because parameters are derived from variables declared globally in the Data Division and almost all the

parameters are in and out parameters. There seems to be no reason, however, for keeping this restriction. The GOM transformation slicing and masking processes, described in Chapter II, work with input/output parameters.

- All functions in the GIM return a single value at the end of their execution and have no output parameters.

Cobol does not implement a function, so all Cobol programs adhere to this restriction.

- All actual parameters in subprogram calls must be variables.

The imperative subprogram calls are built by the translator based on *perform* statements in such a way that all actual parameters are variables.

- Subprograms to be modeled in the GIM are not allowed to make calls to themselves.

Recursion is not allowed in Cobol, either, so legacy Cobol programs satisfy this restriction.

- The call tree of a collection of imperative subprograms must be a directed acyclic graph.

Cobol satisfies the call tree restriction.

- All variables in a subprogram are either declared locally or are formal parameters of the subprogram.

The imperative subprograms are built by the translator based on *perform* statements so that all variables in the subprogram are formal parameters.

- Subprograms cannot be declared inside another subprogram. They are all declared in the main program's global scope.

The imperative subprograms are built based on perform statements so that all subprograms are declared in the main program's global scope.

- The GIM does not model heterogeneous data structures.

As mentioned above, Cobol program makes thorough use of records. Therefore, the transformation system replaces all records with elementary items and transforms the statements that use group items to use the new elementary items. Hence, the legacy program does not need to satisfy this restriction.

- The GIM does not model pointers.

Cobol language does not implement pointers, so the restriction is satisfied.

3.4 Reengineering Methodology.

The methodology for reengineering Cobol programs consists of five phases. In the first phase, the legacy Cobol code is modified by hand to satisfy the restrictions imposed by the GIM and restrictions imposed by the translation system. In the second phase, the program is parsed to generate the input for the transformation system. In the third phase, the Cobol AST is transformed into a new Cobol AST that is more similar to the GIM AST. In the fourth phase the GIM AST is built by the translation system. In the fifth and last phase, the objects are extracted from the GIM and the GOM is built using Sward's prototype system.

The third and fourth phases are based on the Cobol construct classification explained in the previous section. Detailed descriptions of the methodology phases and the complete classification of Cobol constructs are provided in Chapter IV. That chapter

also describes the approach taken to apply the PBOI methodology to a Cobol legacy system.

The program modeled in the GOM can be used to generate a program in an object-oriented language. Research to recover the modeled program modeled into the GOM and to generate the program in an object oriented language are being accomplished at AFIT.

3.5 Methodology Conclusion.

The Cobol language is different from a typical imperative language. Cobol programs are often referred to as being data-intensive [21]. Cobol provides structured data types and almost all its constructs provide multiple operations in just one statement. A Cobol program is heavily record-based, and is allowed two different records to share the same memory locations (redefines clause). In addition, the use of paragraphs and perform statements is not really much like the subprogram calling structure of most imperative languages. Despite the differences between the Cobol AST and the GIM AST, this chapter has provided a description of an overall strategy for the translation of a Cobol program into the GIM.

IV. Design of the Reengineering System

4.1 Overview.

This chapter presents the design of the transformation and translation systems and the overall view of both is shown in Figure 7. The chapter includes the entire classification of the Cobol constructs and the corresponding imperative statements. Restrictions for some Cobol statements are described together with the classification. It also describes how the phases of the PBOI methodology are applied to transform a Cobol legacy system into the GOM.

The transformation system turns the Cobol code into constructs more similar to those of the GIM. Consequently, the translation system has a smaller set of the Cobol constructs as its input. The transformation system output is a Cobol program with constructs that can be translated into the GIM.

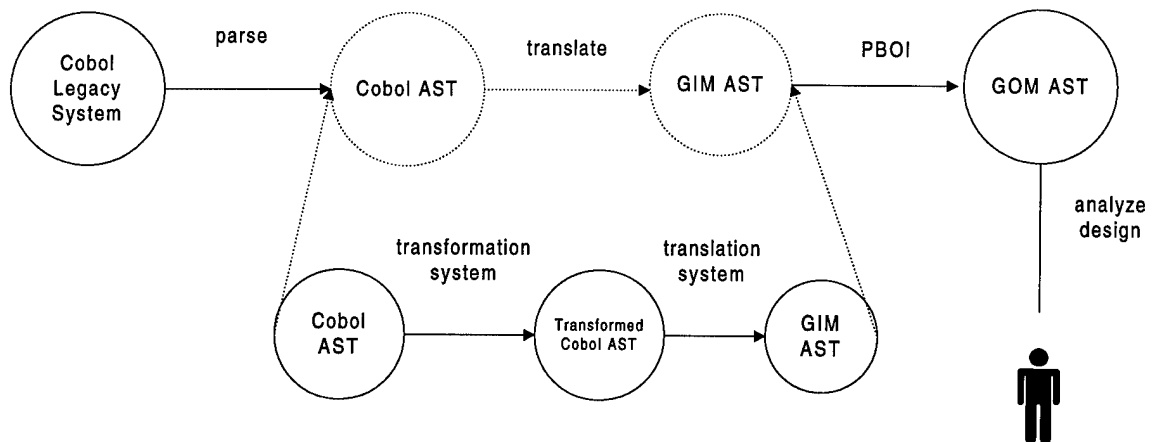


Figure 7 Overall View of Transformation and Translation Systems

The transformation and translation systems are built using Software Refinery that parses in Cobol source code and builds an AST that stores information about the source code. The transformation system builds a new Cobol AST that is more similar to GIM AST. The translation system builds the GIM AST based on the transformed Cobol AST.

4.2 Classification of the Cobol Statements.

The classification phase of the research is responsible for defining the approach used to develop the transformation and translation systems. The four classes used were defined in Chapter III. Table 3 summarizes the classification of the Cobol constructs. The transformable constructs are treated in the transformation system. The directly translatable and indirectly translatable constructs are treated in the translation system.

The constructs that use group items must be treated in the transformation system. The statements were split into several statements, one for each elementary item, and they were renamed with a new identification.

4.3 The Transformation System.

The transformation system in the Cobol reengineering methodology begins with parsing the legacy Cobol program using Refine/Cobol. The parse constructs Cobol AST that is the input for the transformation system. The transformations are applied to the Cobol AST.

The final transformations are responsible for transforming group items. They are final because the group items and their elementary items are necessary to transform the statements.

Table 3 Cobol Construct Classification

Construct Classification	Cobol Construct
Transformable	<p><i>add identifier-1 ... to identifier-2 ... , add identifier-1 ... to identifier-2 giving identifier-3 ...</i> <i>compute identifier-1 identifier-2 = arithmetic-expression</i> <i>display identifier-1 identifier-2 ...</i> <i>divide identifier-1 into identifier-2 ...</i> <i>divide identifier-1 into identifier-2 giving identifier-3 identifier-4 ...</i> <i>move identifier-1 to identifier-2 ...</i> <i>multiply identifier-1 by identifier-2 identifier-3 ...</i> <i>multiply identifier-1 by identifier-2 giving identifier-3 identifier-4 ...</i> <i>perform paragraph-name</i> <i>perform paragraph-name thru end-paragraph-name</i> <i>perform paragraph-name thru paragraph-name identifier-1 times</i> (all statements with group with group item)</p>
Directly Translatable	<p><i>accept , add giving , call , close ,</i> <i>compute identifier = arithmetic-expression ,</i> <i>display identifier,</i> <i>divide identifier-1 into identifier-2 giving identifier-3</i> <i>divide identifier-1 by identifier-2 giving identifier-3</i> <i>if condition-1 , if else/otherwise ,</i> <i>move identifier-1 to identifier-2 ,</i> <i>multiply identifier-1 by identifier-2 giving identifier-3</i> <i>open , read</i> <i>subtract identifier-1 from identifier-2 giving identifier-3</i> <i>write</i></p>
Indirectly Translatable	<p><i>perform varying from by until , perform thru until ,</i> <i>perform thru ,</i></p>
Not Handled	<p><i>cancel , copy , delete , enter , evaluate , exit , generate , goto ,</i> <i>initialize ,</i> <i>inspect , merge , purge , receive , release</i> <i>replace , return , rewrite , search , send</i> <i>set , sort , start , stop run , string ,</i> <i>supress , terminate ,</i> <i>use before reporting , use for debugging</i></p>

4.3.1 Transformable Constructs.

Before each construct transformation explanation, the original Cobol construct is presented with the constructs that are transformed. Also, the restrictions imposed on the constructs are presented.

4.3.1.1 Assignment Transformation.

The *add*, *compute*, *divide*, *move*, *multiply* and *subtract* constructs assign a value to one or more variables. These constructs are not modeled in the GIM but, they can be modeled as imperative assignments. Therefore, these constructs are transformed into several Cobol constructs with just one variable to receive the value of the assignment.

The *add*, *divide*, *multiply* and *subtract* constructs have one format that specifies the variable to receive the assignment value. Therefore, the transformation system converts all kinds of formats to a format using the *giving* clause. The *giving* clause determines the variable that receives the assignment value.

As a result, the transformed Cobol AST is composed with the following format *add*, *compute*, *divide*, *move*, *multiply* and *subtract* constructs.

add identifier-1 ... *giving* identifier-2

compute identifier-1 = arithmetic-expression-1

divide identifier-1 *into* identifier-2 *giving* identifier-3

divide identifier-1 *by* identifier-2 *giving* identifier-3

move identifier-1 *to* identifier-2

multiply identifier-1 *by* identifier-2 *giving* identifier-3

subtract identifier-1 *from* identifier-2 *giving* identifier-3

a. *Add Construct.*

The *add* statement adds two or more data items and assigns the sum value to one or more data items. As the *add* statement allows the variables preceding the *to* clause to be the same as those which receive the result (variables following the *to* clause), there are some concerns in transforming the *add* statement.

1. *add* identifier-1 ... *to* identifier-2 ...

Transformed into several add Cobol statements:

add identifier-1 ... *giving* auxiliary-var

add auxiliary-var-1 *to* identifier-2 *Giving* identifier-2

add auxiliary-var-1 *to* identifier-3 *Giving* identifier-3

...

2. *add* identifier-1 ... *to* identifier-2 *Giving* identifier-3 ...

Transformed into: add and move Cobol statements:

add identifier-1 ... identifier-2 *giving* auxiliary-var

move auxiliary-var *to* identifier-3

move auxiliary-var *to* identifier-4

...

The transformation system creates an auxiliary variable to contain the sum of the left-hand side identifiers (preceding the *to* clause) and a new add statement to add those data items before the clause *to*. Additionally an add statement for each one of the right-

hand side identifiers (following the *to* clause) is created. The new variable holds the sum of the data items. The creation of a new *add* statement and a new variable are necessary to avoid an incorrect assignment. The variables that hold the result can be used as operands on the add statement. The new add statement ensures that the following add statements or move statements are assigned the correct sum value. The new add statements are inserted before the original *add* construct in the statement sequence of the Cobol AST. After the transformations, the auxiliary variables that are created are inserted into the Data Division Working Storage Section. The *add corresponding* statement is also transformed, into several *add* statements, during the group item transformation described in item 4.3.1.4.

The example below shows a Cobol *add* statement and the transformed Cobol *add* statement.

add HEX-1 to 400190-INDEX

Transformed Cobol construct:

add HEX-1 to 400190-INDEX *giving* 400190-INDEX

b. *Compute* Construct.

The compute statement sets one or more data items equal to the value of an arithmetic expression. The *compute* statement with an arithmetic-expression with multiply, divide and power operators is not transformable into the GIM, because the *cache* and *decache* Refine statements used on the transformation and translation systems show problems with these operators. This problem occurs when transforming the statement as follows.

1. *compute* identifier-1 identifier-2 = arithmetic-expression-1

Transformed into several *compute* and *move* Cobol statements:

compute identifier-1 = arithmetic-expression-1

move identifier-1 to identifier-2

...

The *compute* construct is transformed to one *compute* statement and several *move* statements. The result of the compute arithmetic expression is held in the variables before the equal signal. For each variable before the equal signal, except for the first one, a *move* statement is created to move the first variable to the others. The move statements are able to assign the arithmetic expression result to each variable. The move statements are inserted in the statement sequence of Cobol AST after the original compute. After, the original compute is converted to have just the first variable before the equal sign. The example below shows a Cobol *compute* statement and the transformed Cobol *compute* statement.

compute HEX-0, HEX-1 = 400780-INDEX + 1.

Transformed Cobol constructs:

compute HEX-0 = 400780-INDEX + 1.

move HEX-0 to HEX-1.

c. *Divide Construct.*

The *divide* statement divides one data item into one or more such items. Then, the quotient is assigned to one or more data items. The *divide* formats that have phrases to deal with errors, including rounded option and remainder phrases are not transformed.

1. *Divide* identifier-1 *into* identifier-2 ...

Transformed into several *divide* Cobol statements:

Divide identifier-1 *into* identifier-2 *giving* identifier-2

Divide identifier-1 *into* identifier-3 *giving* identifier-3

....

2. *Divide* identifier-1 *into* identifier-2 *giving* identifier-3 identifier-4 ...

Transformed into one *divide* statement and several *move* Cobol statements:

Divide identifier-1 *into* identifier-2 *giving* identifier-3

move identifier-3 *to* identifier-4

...

3. *Divide* identifier-1 *by* identifier-2 *giving* identifier-3 identifier-4 ...

Transformed into one *divide* statement and several *move* Cobol statements:

Divide identifier-1 *by* identifier-2 *giving* identifier-3...

move identifier-3 *to* identifier-4

...

To transform *divide* constructs, it is necessary to create *move* statements to be used in the transformation of *divide* with *giving* clause. It is necessary because one of the variables that hold the result can be used as an operand. So, to avoid an incorrect

assignment, the original *divide* construct is modified to have just the first variable that holds the operation result. The new *move* statements are inserted after the original *divide* in the statement sequence of the Cobol AST. The example below shows a Cobol *divide* statement and the transformed Cobol *divide* statement.

divide DIVIDEND *by* DIVISOR *giving* RESULT1 RESULT2.

Transformed Cobol constructs:

divide DIVIDEND *by* DIVISOR *giving* RESULT1.

move RESULT1 *to* RESULT2.

d. *Move* Construct.

The *move* statement transfers the contents of one data item to one or more other data items. Move statements allow data to be moved from group item to group item. The transformation system restricts the group items involved in a move statement to have the same structure. The *move corresponding* statement is not transformed because records are eliminated in the group item transformation as described in item 4.3.1.4.

1. *move* identifier-1 *to* identifier-2 ...

Transformed into several move Cobol statements:

move identifier-1 *to* identifier-2

move identifier-1 *to* identifier-3

...

A *Move* construct is transformed into several *move* statements. For each variable after the *to* clause, except for the first one, a *move* statement is created. The *move* statements are inserted in the statement sequence of Cobol AST after the original *move* construct. After, the original *move* construct is changed to have just the first variable before the *to* clause. The example below shows a Cobol *move* statement and the transformed Cobol *move* statements.

move HEX-0 , HEX-1 *to* 400190-INDEX.

Transformed Cobol constructs:

move HEX-0 *to* 400190-INDEX.

move HEX-1 *to* 400190-INDEX.

e. *Multiply* Construct.

The multiply statement forms the product of two data items and stores the result in one or more data items. After the transformation, the multiply statement has just one assignment.

1. *Multiply* identifier-1 *by* identifier-2 identifier-3 ...

Transformed into several multiply Cobol statements:

Multiply identifier-1 *by* identifier-2 *giving* identifier-2

Multiply identifier-1 *by* identifier-3 *giving* identifier-3

...

2. *Multiply* identifier-1 *by* identifier-2 *giving* identifier-3 identifier-4 ...

Transformed into one multiply statement and several moves:

Multiply identifier-1 by identifier-2 *giving* identifier-3

Move identifier-3 to identifier-4

...

Like *divide* construct, to transform *multiply* construct it is necessary to create *move* statements to be used in transformation of the *multiply* statement with *giving* clause.

It is necessary, because one of the variables that hold the result can be used as an operand. So, to avoid an incorrect assignment, the original *multiply* construct is modified to have just the first variable that held the operation result. The new *move* statements are inserted after the original divide in the statement sequence of the Cobol AST. The example below shows a Cobol *multiply* statement and the transformed Cobol *multiply* statement.

multiply BASE by RATE1 *giving* RESULT , PERCENTAGE.

Transformed Cobol construct:

multiply BASE by RATE1 *giving* RESULT.

move RESULT to PERCENTAGE.

f. *Subtract* Construct.

The *subtract* statement subtracts a single data item or the sum of two or more data items from one or more data items, and then assigns one or more data items with the result. The *subtract corresponding* is not transformed because the records are eliminated in the group item transformation as described in item 4.3.1.4.

1. *subtract* identifier-1 ... *from* identifier-2 ...

Transformed into add and subtract Cobol statements:

add identifier-1 ... *giving* auxiliary-variable
subtract auxiliary-variable *from* identifier-2 *giving* identifier-2 ,
subtract auxiliary-variable *from* identifier-3 *giving* identifier-3 ,
...

2. *subtract* identifier-1 ... *from* identifier-n *giving* identifier-o identifier-p ...

Transformed into subtract and move Cobol statements:

subtract identifier-1 ... *from* identifier-n *giving* identifier-o
move identifier-o *to* identifier-p
...

To transform the *subtract* construct, without the *giving* clause, it is necessary to create an *add* statement to save the original sum value of the variables before the *from* clause. A new variable is created to hold that sum value. Subtract statements are created, one for each variable after the *from* clause. The new variable is subtracted from each variable after the *from* clause, and the result is saved in the latter variables. For the *subtract* construct with the *giving* clause, the original *subtract* is modified to have just the first variable after the *giving* clause. Also, *move* statements are created to save the result, which is in the first variable, in the other variables after the *giving* clause. The new *add* statement is inserted before the original *subtract* in the statement sequence of the Cobol AST. The example below shows a Cobol *subtract* statement and the transformed Cobol *subtract* statement.

subtract FEDTAXES, STATE-TAXES *from* ITEM-A , ITEM-B.

Transformed Cobol constructs:

add FEDTAXES *to* STATE-TAXES *giving* VAR-AUX.

subtract VAR-AUX *from* ITEM-A *giving* ITEM-A.

subtract VAR-AUX *from* ITEM-B *giving* ITEM-B.

Therefore, the transformed Cobol AST is just built with *add*, *compute*, *divide*, *move*, *multiply* and *subtract* translatable constructs.

These transformations show that to transform a Cobol AST into a GIM AST is not trivial. The Fortran AST has the same assignment statements as the GIM. But, the Cobol does not have explicit assignment statements, and the constructs that can be viewed as assignment statements allow multiple assignments in just one statement.

4.3.1.2 Iterative Control Flow Transformation.

Structured iterative control flow in Cobol is implemented using *perform varying*, *perform time* and *perform until* statements.

Every *perform* statement has its own *thru* clause because there is a previous transformation of all *perform* statements into *perform thru* statements.

The *perform until* is a directly translatable construct and it is directly translated into the GIM. There is no transformation for it.

The *perform varying* is an indirectly translatable construct and it is translated into the GIM. There was no transformation for it.

The *perform time* construct is transformed into a *perform varying* construct. The original *perform time* is converted to a *perform varying* and a new variable is created to control how many times the *perform* statement is executed. Also, the new variable is inserted in the Data Division Working Storage Section. The example below shows a Cobol *perform time* statement and the transformed Cobol *perform time* statement.

perform SUM-OF-ODD-NUMBERS thru END-SUM TOTAL *times*

Transformed Cobol constructs:

perform SUM-OF-ODD-NUMBERS thru END-SUM varying VAR-
LOOP from 1 by 1 until VAR-LOOP = TOTAL.

4.3.1.3 Selective Control Flow Transformation.

The selective control flow in Cobol language is implemented by *if-then-else* and *if-then* statements. The *if* statement is a directly translatable construct. So, this construct is directly translated.

4.3.1.4 Record (Group Item) Transformation/Elimination.

The GIM does not represent records so, group items are eliminated and the elementary items are renamed. Any item in a group item must have a level number numerically greater than that of the group to which it belongs. The statements that use group items have to be altered to use the new data structures. Different group items may have subitems with the same name, to guarantee uniqueness, the elementary items are renamed by joining the old name with the name of the most external group item (in other

words, with the smallest level number group item). The statements subject to the use of the group items are the following: *move* and *display*. The *move* statement has to satisfy one restriction. That is, the group items involved in the operation have the same structure. The *move* statements with group items are transformed into several moves with respective elementary items. The *if* and *perform* statements with group items are not transformed because it is impractical to build the condition expression tree. The transformation implementation restricts record structures so that they have an *occurs* clause on just one level.

Example: 01 400060-PN-CFF occurs 5 times.

05 400070-PN picture X(18).

05 400085-AV picture X(05).

05 400080-CFF picture X(05).

05 400083-PQ picture X(04).

Were transformed into: 01 400070-PN-400060-PN-CFF occurs 5 times picture X(18).

01 400085-AV-400060-PN-CFF occurs 5 times picture X(05).

01 400080-CFF-400060-PN-CFF occurs 5 times picture X(05).

01 400083-PQ-400060-PN-CFF occurs 5 times picture X(04).

The transformation causes effects in the Data and Procedure Divisions. In the Data Division, the group items are converted to elementary items and the elementary items are renamed. In the Procedure Division, the statements that use group items are transformed to use their elementary items and statements using elementary items whose name changes are updated.

The group item transformation is implemented based on a map. The map is built to map a tuple (the most external group item and each of its elementary items) to a new elementary item name.

Transforms are implemented to transform *display* and *move* statements using group items. Based on the map and for each statement (*display* and *move*) that uses elementary items, a transformation renames them using the new elementary item name. Also, the Data Division was traversed. When the group item has an *occurs* clause, an *occurs* clause is created for the elementary items of the group item. The elementary items in the Data Division are renamed with the new elementary item name, and the group items are removed and all level numbers are altered to 1.

For *display* statement using a group item, several *display* statements are created, one for each elementary item in the group item. The new *display* statements are inserted in the same statement sequence where the original *display* is. The example below shows a Cobol *display* statement and the transformed Cobol *display* statement.

display 400680-MSG *upon console*.

Transformed Cobol constructs:

display FILLER-CT-400680-MSG *upon console*.

display FILLER-40-400680-MSG *upon console*.

display 400700-CT-400680-MSG *upon console* .

For a *move* statement using a group item, several *move* statements are created, one for each elementary item in the group item. The new *move* statements are inserted in the

same statement sequence where the original *move* is. A restriction for this transformation is that the group items involved in the operation of the *move* statement must have the same structure. The example below shows a Cobol *move* statement and the transformed Cobol *move* statement.

move '' to 006200-DTL

Transformed Cobol constructs:

move '' to 006215-PN-POS-1-006200-DTL.

move '' to 006230-AV-006200-DTL

move '' 006220-CFF-006200-DTL.

4.3.1.5 Other Transformations.

a. *Display Construct.*

The display statement is used to output the contents of each identifier to a hardware device. Although, the GIM allows multiple outputs because the imperative output list is a sequence.

1. *Display identifier-1 identifier-2 ...*

Transformed into several display Cobol statements:

Display identifier-1

Display identifier-2

...

The example below shows a Cobol *display* statement and the transformed Cobol *display* statement.

display 006220-CFF-006200-DTL *upon* console.

Transformed Cobol construct:

write(CONSOLE , 006220-CFF-006200-DTL)

b. *Perform* Construct.

The *perform* statement executes one or more paragraphs or executes statements that are written within it. A transformation is created to transform a *perform* statement with no *thru* clause into a *perform* statement with a *thru* clause, and to transform a *perform* statement with a *thru* clause into a *perform* statement with a new *thru* paragraph name. After the transformation of the *perform* statement, its meaning changes slightly. Now the new paragraph name (end-paragraph-name) following the *thru* clause delimits the last statement executed by the *perform* statement. In the original meaning the paragraph name following the *thru* clause delimits the last paragraph to be performed. *Perform times* statement is transformed into *perform varying*, so this transformation creates a new variable to control the *varying* clause. After the transformations, the new variable that is created is inserted in the Data Division Working Storage Section. The insertion of the new variable in the Data Division is required because this division is used to transform the variables into the GIM.

1. *perform* paragraph-name

Transformed into perform Cobol statement:

perform paragraph-name *thru* end-paragraph-name

2. *perform* paragraph-name *thru* end-paragraph-name

Transformed into perform Cobol statement:

perform paragraph-name *thru* end_end-paragraph-name

The example below shows a Cobol *perform* statement and the transformed Cobol *perform* statement.

perform SUM-OF-ODD-NUMBERS.

Transformed Cobol construct:

perform SUM-OF-ODD-NUMBERS *thru* END-SUM-OF-ODD-NUMBERS.

This transformation is implemented to make the translation of perform statement into the GIM as an imperative subprogram more direct.

4.3.2 Implementing the Transformation System.

The transformation system's function is to turn a legacy Cobol system into one with more similar constructs to those of the GIM. The output of the transformation system is the input of the translation system.

After parsing the legacy Cobol program, the Cobol AST is traversed in pre-order and for each statement found that matches the left-hand-side of the correspondent

transformation, the right-hand-side of the transformation is built. The traversal begins with the Cobol AST of the entire legacy system. Some transformations in the transformation system transform one construct into several constructs. Therefore, it is necessary to ensure that the new constructs are inserted in the same statement sequences where the original construct is.

Thus, it is necessary to create one Refine transform for each statement sequence attribute in the Cobol AST. The statement sequence attributes subject to have statements are: procedure-sentence-statement-sequence, verb-statement-sequence-1 and verb-statement sequence-2. The following sections describe the transformations that develop.

4.4 The Translation System.

The translation system in the Cobol reengineering methodology begins by traversing the transformed Cobol AST that is the output of the transformation system. The transformations are applied to that Cobol AST.

4.4.1 Directly Translatable Constructs.

The directly translatable constructs are described next. The original Cobol construct is presented with the constructs that are transformed and, the restrictions imposed on the constructs are presented. Also, the variable, data type, expression and input/output translations are described.

a. *Accept* Construct

The *accept* statement transfers data from a hardware device into identifier-1.

1. *accept* identifier-1

Transformed into one read statement.

read(file-name, identifier-1)

b. *Add* Construct

1. *add* identifier-1 ... *giving* identifier-2

Transformed into one assignment imperative statement:

Identifier-2 := identifier-1 +

c. *Call* Construct

The call statement causes control to be transferred from one program to another program.

1. *Call* literal-1 [*using* identifier-1 ...]

Transformed into one subprogram call imperative statement:

Literal-1(identifier-1 ...)

d. *Close* Construct

The close statement terminates the processing of file.

1. *close* file-name-1 ...

Transformed into one close imperative statement:

close file-name-1 ...

e. *Compute Construct*

Compute with an arithmetic-expression with multiply, divide and power operators is not translatable into the GIM. This is because the *cache* and *decache* Refine statements, used on the transformation and translation systems, shows problems with these operators. Thus, the Cobol program cannot have compute construct with an arithmetic-expression that uses divide and power operators.

1. *compute* identifier-1 = arithmetic-expression-1.

Transformed into one assignment imperative statement:

identifier-1 := imperative-expression;

f. *Divide Construct*

1. *divide* identifier-1 *into* identifier-2 *giving* identifier-3

Transformed into one assignment imperative statement:

Identifier-3 := identifier-2 / identifier-1;

2. *divide* identifier-1 *by* identifier-2 *giving* identifier-3

Transformed into one assignment imperative statement:

Identifier-3 := identifier-1 / identifier-2;

g. *If construct*

The *if* statement evaluates a condition and subsequent program action depends on whether the value is true or false.

If statements allow the condition to be a group item, but the transformation system restricts the condition so that it cannot be a group item.

1. *if* condition-1 statement-1...

Transformed into if then else imperative statement:

if condition-1 *then* statement-1...*else null*;

2. *if* condition-1 statement-1...*else* statement-n

Transformed into if then else imperative statement:

if condition-1 *then* statement-1...*else* statement-n ... ;

h. *Move* Construct

Move statements allow data to be moved from group item to group item. The transformation system restricts the group items involved in the operation of the *move* statement the items must have the same structure.

1. *move* identifier-1 *to* identifier-2

Transformed into one assignment imperative statement:

Identifier-2 := identifier-1;

i. *Multiply* Construct

1. *multiply* identifier-1 *by* identifier-2 *giving* identifier-3

Transformed into one assignment imperative statement:

identifier-3 := identifier-1 * identifier-2;

j. *Open Construct*

1. *Open input/output file-name-1*

Transformed into one open imperative statement:

open input/output file-name-1;

k. *Read Construct*

The read statement obtains a record from a file and puts it into the file's record area.

1. *read file-name*

Transformed into one read imperative statement:

read(identifier-file , file-name);

l. *Subtract Construct*

1. *subtract identifier-1 from identifier-2 giving identifier-3*

Transformed into one assignment imperative statement:

Identifier-3 := identifier-2 – identifier-1;

m. *Write Construct*

The write statement writes *record* to a file.

1. write record-name

Transformed into one output imperative statement:

write (file-name , record-name);

n. Variable Translation.

The Cobol variables are declared in the Data Division but the GIM does not have variable declarations.

Therefore, the Cobol variables are translated into the GIM, building an imperative-variable AST and stored in the Imperative Symbol Table.

For each reference to a Cobol variable, an instance of the imperative-name class is built to store scope, identifier and indices information.

o. Data Type Translation.

A Data Description Entry (more specifically a *picture* clause) in the Data Division specifies the characteristics of a data item.

The Cobol category of data items can be either alphabetic, alphanumeric, alphanumeric-edited, numeric or numeric-edited. The occurs clause is used to define a set of repeated data items. The editing characters in the *picture* clause are not used as a format for input/output statements because the GIM does not model editing characters.

Figure 8 shows the transformations to translate the data types.

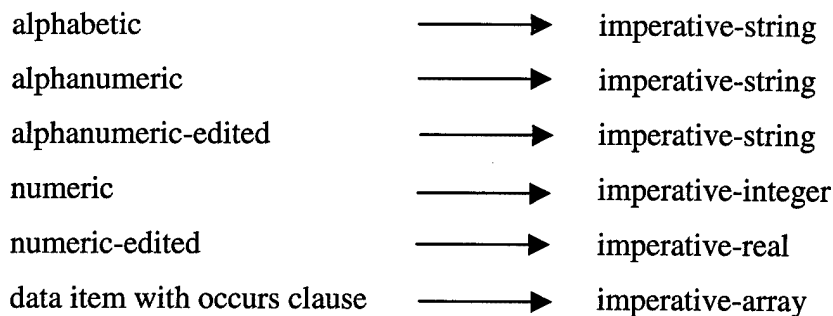


Figure 8 - Imperative Data Type Transformation

p. Imperative Expression Translation.

A Cobol expression can be either an arithmetic expression or a conditional expression. An arithmetic expression can be a single elementary numeric data item and two or more data items or literals connected by arithmetic operator. Figure 9 shows the transformations to translate the arithmetic expression.

add-operator	→	imperative-addition
divide-operator	→	imperative-division
exponentiate-operator	→	imperative-exponent
multiply-operator	→	imperative-multiplication
subtract-operator	→	imperative-subtraction
false-value	→	imperative-literal-false
true-value	→	imperative-literal-true
integer-value	→	imperative-literal-integer
real-value	→	imperative-literal-real
charstring-value	→	imperative-charstring

Figure 9 - Imperative Arithmetic Expression Transformation

A conditional expression is a simple condition or a complex condition. Figure 10 shows the transformations to translate the conditional expression.

and-condition	→	imperative-and
not-condition	→	imperative-not
or-condition	→	imperative-or
equal-operator	→	imperative-equal
greater-than-equal-operator	→	imperative-greater-than-or-equal
greater-than-operator	→	imperative-greater-than
less-than-equal-operator	→	imperative-less-than-or-equal
less-than-operator	→	imperative-less-than

Figure 10 - Imperative Conditional Expression Transformation

q. Input/Output Translation.

The Cobol language implements input by *accept* and *read* statements. Output is implemented by Cobol *display* and *write* statements.

The *accept* and *read* statements are translated into imperative-input and *display* and *write* statements are translated into imperative-output.

The Cobol AST that represents the following write statement

write 006200-DTL.

is translated into the GIM by building one imperative-output. The record name (006200-DTL) is converted to a GIM imp-identifier and stored in the imp-output-list attribute of a GIM imperative output.

The imperative-output is shown below using GIL syntax.

Write(SYS5 , 006200-DTL);

Figure 11 shows the transformation to translate the input/output constructs into imperative input/output.

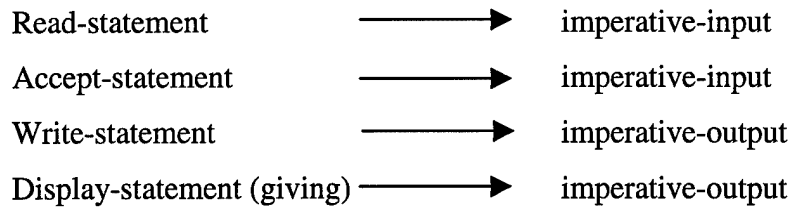


Figure 11 - Imperative Input/Output Transformation

r. Call Translation.

An Imperative subprogram call is implemented in Cobol language by the call statement and, the Cobol perform statement. Therefore, these constructs are translated into the GIM like an imp-subprogram-call AST.

The following Cobol perform and the call statements, are translated into the GIM by building two imp-subprogram-call ASTs.

perform 600010 thru 600030-END.

call 'C18005PA'.

The perform name from the Cobol AST is converted to a GIM variable and stored as the imp-call-identifier of the imp-subprogram-call AST. The sequence of the variables used inside the paragraphs performed by the perform statement are converted to a sequence of GIM variables and stored as the imp-call-actuals parameters in the GIM AST.

The call identifier from the Cobol AST is converted to a GIM variable and stored as the imp-call-identifier of the imp-subprogram-call AST.

The two `imp-subprogram-call` GIM ASTs built from these two translations are shown below using the GIL syntax.

```
600010(..., CHK-UNIF, MODULE-STATUS, ...);  
C18005PA;
```

Figure 12 shows the transformation to translate the constructs into the `imp-subprogram-call`.

<code>perform-statement</code>	→	<code>imp-subprogram-call</code>
<code>call-statement</code>	→	<code>imp-subprogram-call</code>

Figure 12 - Imp-Subprogram-Call Transformation

4.4.2 Indirectly Translatable Constructs.

a. *Perform* Construct

Perform statements allow the condition in the *until* clause to be a group item, but the transformation system restricts the condition so that it cannot be a group item.

Imperative subprograms are implemented in Cobol language by calling another program (a called program).

A *perform* statement has a similar function to a program call. Therefore, the code between the first paragraph and the last one performed by the *perform* statement is considered a subprogram.

The variables used inside the performed paragraphs are treated like parameters.

The performed paragraphs before the *stop run* statement are translated into the GIM as subprograms and also they are kept inside the main program. The main program is identified as the code between the first statement in the Procedure Division until the last statement before the *stop run* statement.

The performed paragraphs after *stop run* are translated into the GIM as subprograms, but in this case, there is no code duplication.

The following Cobol perform statement, and the corresponding performed paragraphs are translated into the GIM by building an imperative-procedure AST.

```
PROCEDURE DIVISION.  
  perform 600010 thru END-600030-END.  
  ...  
600010.  
  display 'Create the Reduced Master File P-300' upon console.  
  ...  
600020.  
  if CHK-UNIF not = 00  
    display 'Open Error Unif-Ckh = ' CHK-UNIF  
    move ' ' to MODULE-STATUS  
    otherwise  
      move ' ' to 006200-DTL.  
  ...  
600030.  
  move CURRENT-DATE to 400790-DATA-RESP.  
END-600030-END.
```

The perform name (600010) is converted into a GIM variable and stored as the imp-subprog-identifier of the imperative-subprogram AST. The variables (CHK-UNIF,MODULE-STATUS,...) used inside the paragraphs (600010, 600020 and 600030) are retrieved from a Refine map, converted into GIM variables and stored in the sequence

of imp-subprog-formals parameters for the GIM AST. Each statement from the performed paragraphs is converted into a GIM statement and stored in the sequence of statements for the imperative-procedure AST.

The imperative-procedure AST is shown below using the GIL syntax.

```
Procedure 600010(...,CHK-UNIF,MODULE-STATUS,...)
Begin
  write (SYS5 , 'Gerar os Mestres Reduzidos P-300');
  if CHK-UNIV not = 00 then
    write(SYS5,' Erro abertura Unif Ckh = ');
    write(SYS5,CHK-UNIF);
  else
    006200-DTL := ' ';
  end if;
end;
```

1. *perform* paragraph-name *thru* end-paragraph-name *until* condition-1

Transformed into one while imperative statement:

```
while not condition-1 do
```

```
  Paragraph-name(all variables used in the paragraphs executed by the
                    perform statement);
```

```
end-while;
```

2. *perform* paragraph-name *thru* end-paragraph-name

Transformed into one subprogram call imperative statement:

```
paragraph-name(all variables used in the paragraphs executed by the
                perform statement);
```

3. *perform* paragraph-name *thru* end-paragraph-name *varying* identifier-1 *from* identifier-2 *by* identifier-3 *until* condition-1

Transformed into one assignment and while imperative statement:

Identifier-1 := identifier-2;

While not condition-1 *do*

Paragraph-name(all variables used in the paragraphs executed by the
perform statement);

end-while;

4.4.3 Constructs Not Handled.

The Cobol constructs are summarized in Table 3, which also show the not-handle constructs that are not implemented into the GIM. These constructs do not have equivalent GIM constructs. The *evaluate* Cobol construct determines the value of one or more conditions and subsequent program action depends on the result. Therefore, the *evaluate* construct can be transformed into the GIM to an *if-then-else* Cobol statement. This transformation is not implemented, because *evaluate* construct is a new feature of 85 Cobol and it is not usually found in legacy Cobol systems. The *stop run* construct is not transformed into the GIM, but it is used to determine the main program boundary.

4.4.4 Implementing the Translation System.

The translation system's function is to translate a Cobol program in canonical form into the GIM. The input of the translation system is the output of the transformation system. Table 4 shows the constructs that the translation system translates into the GIM.

The Cobol AST is traversed in pre-order and, for each perform-statement found a map is created to relate the perform paragraph-name to its statements and its variables.

Table 4 Cobol Constructs Recognized by the Translation System

<i>Accept</i> identifier-1
<i>Add</i> identifier ... <i>giving</i> identifier-n
<i>Call</i> literal
<i>Call</i> literal using identifier ...
<i>Close</i> file-name ...
<i>Compute</i> identifier = arithmetic expression
<i>Display</i> identifier
<i>Divide</i> identifier-1 <i>into</i> identifier-2 <i>giving</i> identifier-3
<i>Divide</i> identifier-1 <i>by</i> identifier-2 <i>giving</i> identifier-3
<i>If</i> condition statement-1
<i>Move</i> identifier-1 <i>to</i> identifier-2
<i>Multiply</i> identifier-1 <i>by</i> identifier-2 <i>giving</i> identifier-3
<i>Open input</i> file-name
<i>Open output</i> file-name
<i>Perform</i> paragraph-name <i>thru</i> end-paragraph-name
<i>Perform</i> paragraph-name <i>thru</i> end-paragraph-name <i>until</i> condition
<i>Perform</i> paragraph-name <i>thru</i> end-paragraph-name <i>varying</i> identifier-1 <i>from</i> identifier-2 <i>by</i> identifier-3 <i>until</i> condition
<i>Read</i> file-name
<i>Subtract</i> identifier-1 <i>from</i> identifier-2 <i>giving</i> identifier-3
<i>Write</i> record-name

The translations use some maps to facilitate the transformations. During the translation it is necessary to have information about the Data Division or other AST objects. Therefore, the information is retrieved from the maps that are constructed before the translation.

The Expression-Table and Conditional-Table maps are constructed to identify the operators and operands in a Cobol expression.

The Expression-Table maps a Cobol *arithmetic-expression* to a sequence of Cobol *arithmetic-expression*. It is necessary to map each arithmetic operator and its operands.

The Conditional-Table maps a Cobol-Object to a sequence of Cobol expression. It is necessary to map each conditional operator and its operands.

The Fake-Symbol-Table is constructed to map each *perform* statement to a sequence of *data-description-entry* that is used in the paragraphs executed by the *perform* statement.

The Statement-Table is constructed to map each *perform* statement to the statements executed by the *perform* statement.

The All-Parameters map is constructed to map each *perform* statement to the *data-description-entry* used in the paragraphs executed by the *perform* statement and the other *data-description-entry* used in the paragraphs executed by any *perform* inside the first *perform*.

4.4.4.1 Imperative Main Program Translation.

The main program is identified as starting at the first statement in the Procedure Division and stopping at the last statement before the stop run statement. The Cobol AST

tree is traversed and for each statement found, the sequence of imperative-program-construct (imp-subprog-statements attribute) is appended with the statement.

4.5 Modifications to the PBOI Prototype.

The PBOI prototype had to be modified to satisfy the new release of Refine software and the new aspects of the Brazilian Air Force Cobol legacy system.

The modification needed because of the new version of Refine was to change the rule check-delta-get and check-delta-set to use the *replace x by* statement. These rules are responsible for exchanging the variables that are class attributes with get and set methods.

The PBOI prototype contains some hard-coded details specific to the BMDSIM Fortran system [19]. Therefore, the PBOI prototype has to be modified to deal with the Cobol system.

The specific modifications are:

1. To alter the directory names in the imp-reload.re and gom-save-pob.re files;
2. To initialize the variable *main-program* in the gim-methods.re file with the main program name;
3. To assign the variable sequence *user-def-subs* in the gim-methods.re file with all the subprogram names of the legacy system(this sequence must also have the subprogram names that are generated during the slicing process);
4. To assign an integer to each subprogram in the imp-reload.re file.

The subprograms called by the main program are transformed before the main one is. The PBOI system uses inter-procedural slicing [20] to build a program slice from a subprogram. The first step is converting the GIM into the GOM is to slice the GIM AST. As the PBOI system uses inter-procedural slicing[20], it is required that the slicing process start in the subprograms that appear at the leaf level of the call tree of the generic imperative design. This step is accomplished with the test-test-check-subp-calls function.

The entire transformation to convert the GIM into the GOM is accomplished by:

1. Running all the program slicing system files, loading the entire legacy system and selecting the auto load slicing and auto load for C1AD99T1;
2. Setting the transformation focus on the main program;
3. Verifying the subprogram category classification with the test-classify function;
4. Slicing each subprogram category 4 and 5 and the main program with the test-test-check-subp-calls function;
5. Checking the results of each slicing process with the test-check-inter-complete function;
6. Masking all the other output parameters other than the slice variable to local variables with test-mask-all-others function;
7. Loading the transformation system with the make-system “~srodrigu/research/prototype/transform”;
8. Choosing the auto load slices, auto load form C1AD99T1 (the main program) auto load saved designs, auto saved designs and C1AD99T1, load all options; and
9. Focusing on the subprograms in the leaf program (of the system call diagram) to perform the sigma(1, 2 or 3) option in the transformation menu;

10. Merging the overlapping classes(manually) from the *current-ood* (object-oriented design).

The slicing process converts the category 4 subprogram into multiple category 2 subprograms, and converts the category 5 subprogram into either multiple category 2 or category 3 subprograms.

After each slicing, it is necessary to check if the called subprograms are still category 4 or 5. This step is accomplished with the test-check-inter-complete. For each subprogram that is still category 4 or 5, the masking process has to be run.

The second step to convert the GIM into the GOM is the masking process. The masking process is accomplished by running the test-mask-all-others function for each variable to be masked in the subprogram.

Therefore, additional knowledge is to know (after slicing), what category each subprogram is.

The sigma transformation process should be automatic because the user should be able to simply select the system root. However, the PBOI prototype does not work well because it run indefinitely and does not produce any classes. Finally, the merging process is accomplished by running the test-test-trans-merge-overlap function.

4.6 Summary.

This chapter has presented the methodology development used to construct the transformation and translation system and how to run the PBOI prototype. The classification of the Cobol constructs has been presented and the restrictions applied to

each construct have been described. The transformations applied to translate specific Cobol constructs into GIM AST have also been described.

Table 5 shows a summary of the Cobol constructs and their corresponding GIM constructs.

Table 5 Cobol Constructs X GIM Constructs

COBOL CONSTRUCT	GIM CONSTRUCT
<i>Accept</i> identifier-1	read(identifier-file , file-name)
<i>add</i> identifier-1 ... <i>giving</i> identifier-n	identifier-n := identifier-1 + ...
<i>call</i> literal-1	literal-1
<i>call</i> literal-1 using identifier-1 ...	literal-1(identifier-1,)
<i>Close</i> file-name-1 ...	<i>close</i> file-name-1
<i>Compute</i> identifier-1 = arithmetic expression	identifier-1 := arithmetic-expression
<i>Display</i> identifier-1	<i>write</i> (file-name,identifier-1)
<i>Divide</i> identifier-1 <i>into</i> identifier-2 <i>giving</i> identifier-3	identifier-3 := identifier-2 / identifier-1
<i>Divide</i> identifier-1 <i>by</i> identifier-2 <i>giving</i> identifier-3	identifier-3 := identifier-1 / identifier-2
<i>if</i> condition statement-1	<i>if</i> condition then statement-1 ... <i>else</i> null end if
<i>if</i> condition statement-1 <i>Otherwise</i> statement-n	<i>if</i> condition then statement-1 ... <i>else</i> statement-n end if
<i>Move</i> identifier-1 <i>to</i> identifier-2	identifier-2 := identifier-1
<i>Multiply</i> identifier-1 <i>by</i> identifier-2 <i>giving</i> identifier-3	identifier-3 := identifier-1 * identifier-2

<i>Open input</i> file-name	<i>open</i> file-name
<i>Open output</i> file-name	<i>open</i> file-name
<i>Perform</i> paragraph-name <i>thru</i> end-paragraph-name	paragraph-name(actual parameters)
<i>Perform</i> paragraph-name <i>thru</i> end-paragraph-name <i>until</i> condition	while not condition do paragraph-name(actual parameters) end do
<i>Perform</i> paragraph-name <i>thru</i> end-paragraph-name <i>varying</i> identifier-1 <i>from</i> identifier-2 <i>by</i> identifier-3 <i>until</i> condition	identifier-1 := identifier-2 <i>while</i> not condition <i>do</i> paragraph-name(actual parameters); identifier-1 := identifier-1 + identifier-3; <i>end do</i>
<i>Read</i> file-name	<i>read</i> (identifier-file, file-name)
<i>Subtract</i> identifier-1 <i>from</i> identifier-2 <i>giving</i> identifier-3	identifier-3 := identifier-1 – identifier-2
<i>Write</i> record-name	<i>write</i> (file-name , record-name)

V. Analysis of the Methodology Applied to a FAB Cobol Legacy System

5.1 The Brazilian Air Force Cobol Legacy System Transformation

The Cobol system selected to undergo the reengineering process was brought from the Air Force in Brazil. This system is part of the 300 project. This project is responsible for controlling the maintenance of the military aircraft. This system was developed on October 2 1969, and from that time until now it has undergone maintenance to assist client needs, thereby making it more and more complex. Appendix A shows the legacy Cobol program that was selected.

5.2 Converting Cobol System to the GIM.

The original system possessed GO TO statements that were removed to make the system compatible with the GIM. The GO TO statements were structured, and they were removed easily from the program. The statements were replaced by *if* statements or by repeating small sections of the code.

The Brazilian Air Force Cobol legacy system C1AD99T1 included a main program which had 39 paragraphs and a total of 304 lines in the Procedure Division. Appendix A shows the legacy Cobol code used for the translation into the GIM.

The system was parsed using the Refine/Cobol and the Cobol AST was traversed. The transformation system generated the Cobol legacy system with constructs more similar to the GIM constructs. After, the translation system transformed the C1AD99T1 system into the GIM.

The translation of the Cobol legacy system into the GIM took eleven minutes. After the Cobol system was transformed into the GIM, the system included the main program, 19 subprograms and a total of 563 lines. Appendix B shows the imperative code using the Generic Imperative Language (GIL) after the translation of the legacy system into the GIM.

Almost all the subprograms were category 5 subprograms producing many output parameters.

5.3 Converting GIM to the GOM.

The last phase in the Cobol reengineering methodology is to execute the system that implements PBOI to extract the objects and to store them into GOM.

The GOM and PBOI were described in chapter I, and detailed information about GOM and PBOI can be found in the Sward's dissertation [22].

The PBOI input is the GIM AST that is saved as Persistent Object Base (POB) file after the translation of the Cobol program. POB file is a group of objects as a Unix file. This is a Refine capability and the file can be saved and loaded in a subsequent session to recreate the group of objects. The PBOI output is the GOM AST.

The test-classify function, responsible for verifying the subprogram category classification, identified a subprogram that had the same output parameter as the left-hand side of different assignment statements as a category 4 or 5, although it should have identified it as category 2 or 3. After the slicing and masking process, that function classified some sliced subprograms incorrectly. The wrong subprogram classifications were written within parentheses in Table 7.

A hidden GOM restriction is that the subprogram names that must be in the variable sequence *user-def-subs* in the PBOI prototype cannot begin with numbers. It is required that the subprogram names begin with an alpha character.

Before running the PBOI with the C1AD99T1 system, a piece of it was used to determine how the PBOI prototype would function. Using this sample with the main program and four subprograms, two category 4, and two category 5 subprograms, the slicing process took about three hours and the sigma transformations took more than eleven. So, transforming the entire system would have been impractical, because almost all the subprograms produced many output parameters, and that would have generated many sliced programs. As a result, the C1AD99T1 system was reduced to make the transformation of the system into the GOM viable. Eight paragraphs that generated eight category 5 subprograms were eliminated from the system. These eliminations did not affect the meaning of the system greatly, because they resulted in the elimination of some groups of records that were to be processed.

Therefore, the system was reduced to one main program and 19 subprograms with different categories (as can be seen in Figure 13 and Table 6).

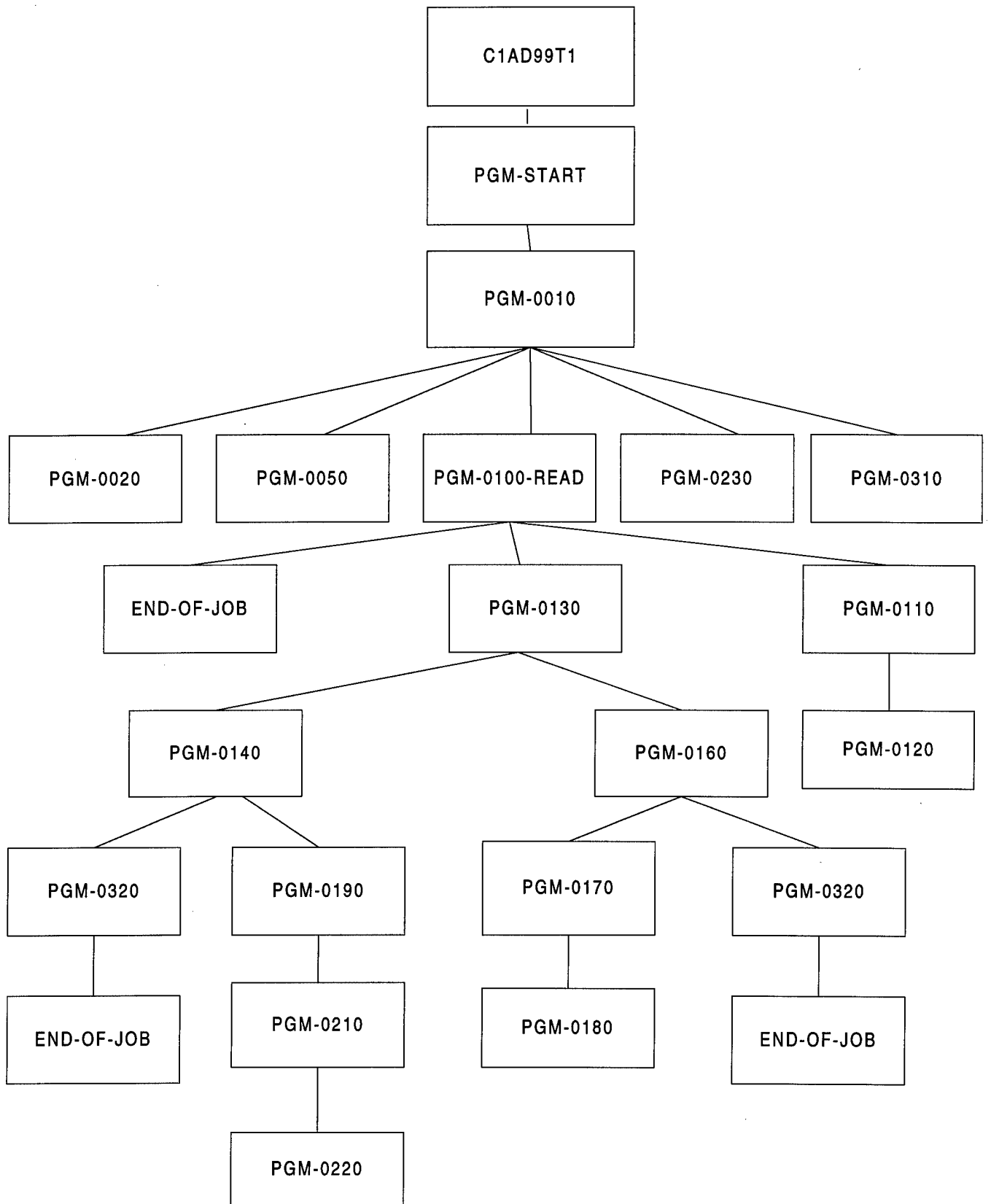


Figure 13 - System Diagram

Table 6 – Category Subprograms and Produced Output

Subprogram(performed paragraph)	Cat.	Data Items Produced in the imperative model
C1AD99T1	1	
PGM-START	5	006215-PN-POS-1-006200-DTL
		006220-CFF-006200-DTL
		006230-AV-006200-DTL
		006246-BL-006200-DTL
		006250-NOMEN-006200-DTL
		006253-UN-006200-DTL
		006255-CAT-006200-DTL
		006260-OA-006200-DTL
		006263-APL-006200-DTL
		006265-TPR-006200-DTL
		006270-FRG-006200-DTL
		006280-TRG-006200-DTL
		006285-RECUP-POR-006200-DTL
		006287-CON-006200-DTL
		006290-ESTOQUE-006200-DTL
		006300-EC-006200-DTL
		006310-OS-006200-DTL
		006320-REP-006200-DTL
		006330-AVG-PRICE-006200-DTL
		006350-A-006200-DTL
		006360-SHELF-006200-DTL
		006229-LOC-006200-DTL
		006375-LAST-ACQ-PRICE-006200-DTL
		006376-PROC-IN-REWORK-006200-DTL
		006377-COND-IN-REWORK-006200-DTL
		006380-SUPERADOR-006200-DTL
		006390-SUPERADO-006200-DTL
		006400-ALTERADO-006200-DTL
		006430-PRE-CALC-006200-DTL
		006440-NMAX-CALC-006200-DTL
		006450-CON-TOTAL-006200-DTL
		006470-MES-RECEB-006200-DTL
		006480-ANO-RECEB-006200-DTL
		006481-Q-P-ART-006200-DTL
		006482-Q-COMPRADA-006200-DTL
		006510-CTL-006200-DTL
		006520-TRAILER-ID-006200-DTL
		006530-RCDS-006200-DTL
		400033-LOC-400010-TABLE
		400080-CFF-400050-PN-CFF

		400083-PQ-400050-PN-CFF
		400085-AV-400050-PN-CFF
		400263-BOMBA-400260-BOMBA
		400266-BOMBA-400260-BOMBA
		400300-C-400280-9-REC
		400530-LOC-400510-ID
		450030-X-SPACE-001100-MASTER-0
		450040-PART-NO-001100-MASTER-0
		450100-FED-MFG-CDE-001100-MASTER-0
		400070-PN-400050-PN-CFF
		400115-DAY-400110-DATE
		400120-ME-400110-DATE
		400130-AN-400110-DATE
		400155-DAY-400140-HOLD
		400160-ME-400140-HOLD
		400170-AN-400140-HOLD
		400700-CT-400680-MSG
		400740-DATE
		400780-INDEX
		400800-D-400790-DATA-RESP
		400820-M-400790-DATA-RESP
		400840-A-400790-DATA-RESP
		SWITCH-0130-PATH-CONTROL-SWITCHES
		VAR-AUX
		MODULE-STATUS-MODULE-ACTIVATION-CONTROL
		400100-POS-40090-RESPONSE
		400185-SWT-400180-TEST
		400550-AV-400510-ID
		400036-AV-400010-TABLE
		400210-0-CT
		400190-INDEX
PGM-0010	5	MODULE-STATUS-MODULE-ACTIVATION-CONTROL
		400100-POS-40090-RESPONSE
		400190-INDEX
		400550-AV-400510-ID
		400036-AV-400010-TABLE
		400530-LOC-400510-ID
		400185-SWT-400180-TEST
		400210-0-CT
		VAR-AUX

		SWITCH-0130-PATH-CONTROL-SWITCHES
		400263-BOMBA-400260-BOMBA
		400266-BOMBA-400260-BOMBA
		450040-PART-NO-001100-MASTER-0
		400780-INDEX
		006530-RCDS-006200-DTL
		400700-CT-400680-MSG
		400070-PN-400050-PN-CFF
		400033-LOC-400010-TABLE
		400080-CFF-400050-PN-CFF
		450100-FED-MFG-CDE-001100-MASTER-0
		400085-AV-400050-PN-CFF
		450030-X-SPACE-001100-MASTER-0
		400300-C-400280-9-REC
		400083-PQ-400050-PN-CFF
PGM-0020	4	400100-POS-40090-RESPONSE
		400185-SWT-400180-TEST
		400190-INDEX
		MODULE-STATUS-MODULE-ACTIVATION-CONTROL
PGM-0050	4	400190-INDEX
		400530-LOC-400510-ID
		400550-AV-400510-ID
		MODULE-STATUS-MODULE-ACTIVATION-CONTROL
PGM-0100-READ	5	400210-0-CT
		MODULE-STATUS-MODULE-ACTIVATION-CONTROL
		400033-LOC-400010-TABLE
		400036-AV-400010-TABLE
		400070-PN-400050-PN-CFF
		400080-CFF-400050-PN-CFF
		400083-PQ-400050-PN-CFF
		400085-AV-400050-PN-CFF
		400210-0-CT
		400263-BOMBA-400260-BOMBA
		400300-C-400280-9-RECT
		400266-BOMBA-400260-BOMBA
		400700-CT-400680-MSG
		400780-INDEX
		450030-X-SPACE-001100-MASTER-0
		450040-PART-NO-001100-MASTER-0

		450100-FED-MFG-CDE-001100-MASTER-0
		MODULE-STATUS-MODULE-ACTIVATION-CONTROL
		SWITCH-0130-PATH-CONTROL-SWITCHES,
		VAR-AUX,
PGM-0110	5	400070-PN-400050-PN-CFF
		400080-CFF-400050-PN-CFF
		400083-PQ-400050-PN-CFF
		400085-AV-400050-PN-CFF
PGM-0120	4	400070-PN-400050-PN-CFF
		400080-CFF-400050-PN-CFF
		400083-PQ-400050-PN-CFF
		400085-AV-400050-PN-CFF
PGM-0130	5	SWITCH-0130-PATH-CONTROL-SWITCHES
		400263-BOMBA-400260-BOMBA
		400266-BOMBA-400260-BOMBA
		400033-LOC-400010-TABLE
		400780-INDEX
		MODULE-STATUS-MODULE-ACTIVATION-CONTROL
		400036-AV-400010-TABLE
		006530-RCDS-006200-DTL
		400700-CT-400680-MSG
		VAR-AUX,
		400070-PN-400050-PN-CFF
		400080-CFF-400050-PN-CFF
		400083-PQ-400050-PN-CFF
		400085-AV-400050-PN-CFF
PGM-0140	5	400263-BOMBA-400260-BOMBA
		400266-BOMBA-400260-BOMBA
		SWITCH-0130-PATH-CONTROL-SWITCHES
		400033-LOC-400010-TABLE
		400780-INDEX
		MODULE-STATUS-MODULE-ACTIVATION-CONTROL
		400036-AV-400010-TABLE
		006530-RCDS-006200-DTL
		400700-CT-400680-MSG
		VAR-AUX

PGM-0160	5	400263-BOMBA-400260-BOMBA
		400266-BOMBA-400260-BOMBA
		400070-PN-400050-PN-CFF
		400080-CFF-400050-PN-CFF
		400083-PQ-400050-PN-CFF
		400085-AV-400050-PN-CFF
		400300-C-400280-9-REC
		400263-BOMBA-400260-BOMBA
		400266-BOMBA-400260-BOMBA
		MODULE-STATUS-MODULE- ACTIVATION-CONTROL
		006530-RCDS-006200-DTL
		400700-CT-400680-MSG
		VAR-AUX
PGM-0170	5	400070-PN-400050-PN-CFF
		400080-CFF-400050-PN-CFF
		400083-PQ-400050-PN-CFF
		400085-AV-400050-PN-CFF
PGM-0180	4	400300-C-400280-9-REC
		MODULE-STATUS-MODULE- ACTIVATION-CONTROL
PGM-0190	5	400033-LOC-400010-TABLE
		400036-AV-400010-TABLE
		400780-INDEX
		MODULE-STATUS-MODULE- ACTIVATION-CONTROL
PGM-0210	5	400780-INDEX
		MODULE-STATUS-MODULE- ACTIVATION-CONTROL
		400036-AV-400010-TABLE
PGM-0220	2	400036-AV-400010-TABLE
PGM-0230	2	MODULE-STATUS-MODULE- ACTIVATION-CONTROL
PGM-0310	2	MODULE-STATUS-MODULE- ACTIVATION-CONTROL
PGM-0320	5	VAR-AUX
		006530-RCDS-006200-DTL
		400266-BOMBA-400260-BOMBA
		400700-CT-400680-MSG
END-OF-JOB	2	VAR-AUX

The process of slicing and masking took more than 51 hours. The 19 subprograms generated 180 slices. The number of sliced programs was so large, because the subprogram generated many output parameters. Table 7 shows the sliced subprograms and their categories.

Next, there was an attempt to generate the classes from the sliced subprogram using the sigma option in the PBOI prototype. This process should have been automatic but it did not work well. Instead the process was applied manually, and for each subprogram the corresponding sigma transformation was performed. From bottom, 65 subprograms were converted into classes. This manual process took more than 84 hours, and it did not work well.

Table 7 Sliced Subprograms

Subprogram	Cat	Slices	Cat	Masked
PGM-0010	5	PGM-0010-400100-POS-40090-RESPONSE	3	X
		PGM-0010-400185-SWT-400180-TEST	3	X
		PGM-0010-400190-INDEX	3	X
		PGM-0010-MODULE-STATUS-MODULE-ACTIVATION-CONTROL	3	X
		PGM-0010-400070-PN-400050-PN-CFF	3	X
		PGM-0010-400080-CFF-400050-PN-CFF	3	X
		PGM-0010-400083-PQ-400050-PN-CFF	3	X
		PGM-0010-400085-AV-400050-PN-CFF	3	X
		PGM-0010-400550-AV-400510-ID	3	X
		PGM-0010-400036-AV-400010-TABLE	3	X
		PGM-0010-400530-LOC-400510-ID	3	X
		PGM-0010-400033-LOC-400010-TABLE	3	X
		PGM-0010-400210-0-CT	3	X
		PGM-0010-VAR-AUX	3	X
		PGM-0010-SWITCH-0130-PATH-CONTROL-SWITCHES	3	X
		PGM-0010-400263-BOMBA-400260-BOMBA	3	X
PGM-0010-400266-BOMBA-400260-BOMBA	3	X		

		PGM-0010-450040-PART-NO-001100-MASTER-0	3	X
		PGM-0010-400780-INDEX	3	X
		PGM-0010-006530-RCDS-006500-TRLR	3	X
		PGM-0010-400700-CT-400680-MSG	3	X
		PGM-0010-450100-FED-MFG-CDE-001100-MASTER-0	3	X
		PGM-0010-450030-X-SPACE-001100-MASTER-0	3	X
		PGM-0010-400300-C-400280-9-REC	3	X
PGM-0020	4	PGM-0020-400100-POS-400090-RESPONSE	2	X
		PGM-0020-400185-SWT-400180-TESTE	2	X
		PGM-0020-400190-INDEX	2	X
		PGM-0020-MODULE-STATUS-MODULE-ACTIVATION-CONTROL	2	X
PGM-0050	4	PGM-0050-400190-INDEX	2	X
		PGM-0050-400530-LOC-400510-ID	2	
		PGM-0050-400550-AV-400510-ID	2	
		PGM-0050-MODULE-STATUS-MODULE-ACTIVATION-CONTROL	2	X
PGM-0100-READ	5	PGM-0100-READ-006530-RCDS-006500-TRLR	3	X
		PGM-0100-READ-400033-LOC-400010-TABLE	3	X
		PGM-0100-READ-400036-AV-400010-TABLE	3	X
		PGM-0100-READ-400070-PN-400050-PN-CFF	3	X
		PGM-0100-READ-400080-CFF-400050-PN-CFF	3	X
		PGM-0100-READ-400083-PQ-400050-PN-CFF	3	X
		PGM-0100-READ-400085-AV-400050-PN-CFF	3	X
		PGM-0100-READ-400210-0-CT	2(3)	X
		PGM-0100-READ-400263-BOMBA-400260-BOMBA	3	X
		PGM-0100-READ-400266-BOMBA-400260-BOMBA	3	X
		PGM-0100-READ-400300-C-400280-9-RECT	3	X
		PGM-0100-READ-400700-CT-400680-MSG	3	X
		PGM-0100-READ-400780-INDEX	3	X
		PGM-0100-READ-450030-X-SPACE-001100-MASTER-0	2(3)	

		PGM-0100-READ-450040-PART-NO-001100-MASTER-0	2(3)	
		PGM-0100-READ-450100-FED-MFG-CDE-001100-MASTER-0	2(3)	
		PGM-0100-READ-MODULE-STATUS-MODULE-ACTIVATION-CONTROL	3	X
		PGM-0100-READ-SWITCH-0130-PATH-CONTROL-SWITCHES	3	X
		PGM-0100-READ-VAR-AUX	3	X
PGM-0110	5	PGM-0110-400070-PN-400050-PN-CFF	3	
		PGM-0110-400085-AV-400050-PN-CFF	3	
		PGM-0110-400080-CFF-400050-PN-CFF	3	
		PGM-0110-400083-PQ-400050-PN-CFF	3	
PGM-0120	4	PGM-0120-400070-PN-400050-PN-CFF	2	
		PGM-0120-400085-AV-400050-PN-CFF	2	
		PGM-0120-400080-CFF-400050-PN-CFF	2	
		PGM-0120-400083-PQ-400050-PN-CFF	2	
PGM-0130	5	PGM-0130-SWITCH-0130-PATH-CONTROL-SWITCHES	3	
		PGM-0130-400263-BOMBA-400260-BOMBA	3	X
		PGM-0130-400266-BOMBA-400260-BOMBA	3	X
		PGM-0130-400033-LOC-400010-TABLE	3	X
		PGM-0130-400780-INDEX	3	X
		PGM-0130-MODULE-STATUS-MODULE-ACTIVATION-CONTROL	3	X
		PGM-0130-400036-AV-400010-TABLE	3	X
		PGM-0130-006530-RCDS-006500-TRLR	3	X
		PGM-0130-400700-CT-400680-MSG	3	X
		PGM-0130-VAR-AUX	3	X
		PGM-0130-400070-PN-400050-PN-CFF	3	X
		PGM-0130-400080-CFF-400050-PN-CFF	3	X
		PGM-0130-400083-PQ-400050-PN-CFF	3	X
		PGM-0130-400085-AV-400050-PN-CFF	3	X
		PGM-0130-400300-C-400280-9-REC	3	X

PGM-0140	5	PGM-0140-SWITCH-0130-PATH-CONTROL-SWITCHES	2(3)	
		PGM-0140-400263-BOMBA-400260-BOMBA	2(3)	
		PGM-0140-400266-BOMBA-400260-BOMBA	3	X
		PGM-0140-400033-LOC-400010-TABLE	3	X
		PGM-0140-400780-INDEX	3	X
		PGM-0140-MODULE-STATUS-MODULE-ACTIVATION-CONTROL	3	
		PGM-0140-400036-AV-400010-TABLE	3	X
		PGM-0140-006530-RCDS-006500-TRLR	3	X
		PGM-0140-400700-CT-400680-MSG	3	X
		PGM-0140-VAR-AUX	3	
PGM-0160	5	PGM-0160-400070-PN-400050-PN-CFF	3	X
		PGM-0160-400263-BOMBA-400260-BOMBA	3	X
		PGM-0160-400266-BOMBA-400260-BOMBA	3	X
		PGM-0160-400080-CFF-400050-PN-CFF	3	X
		PGM-0160-400083-PQ-400050-PN-CFF	3	X
		PGM-0160-400085-AV-400050-PN-CFF	3	X
		PGM-0160-400300-C-400280-9-REC	3	X
		PGM-0160-MODULE-STATUS-MODULE-ACTIVATION-CONTROL	3	X
		PGM-0160-006530-RCDS-006500-TRLR	3	X
		PGM-0160-400700-CT-400680-MSG	3	X
		PGM-0160-VAR-AUX	3	X
PGM-0170	5	PGM-0170-400070-PN-400050-PN-CFF	2(3)	
		PGM-0170-400080-CFF-400050-PN-CFF	2(3)	
		PGM-0170-400083-PQ-400050-PN-CFF	2(3)	
		PGM-0170-400085-AV-400050-PN-CFF	2(3)	
		PGM-0170-400300-C-400280-9-REC	3	
		PGM-0170-MODULE-STATUS-MODULE-ACTIVATION-CONTROL	3	
PGM-0180	4	PGM-0180-400300-C-400280-9-REC	2	
		PGM-0180-MODULE-STATUS-MODULE-ACTIVATION-CONTROL	2	
PGM-0190	5	PGM-0190-400033-LOC-400010-TABLE	2(3)	
		PGM-0190-400780-INDEX	3	X
		PGM-0190-MODULE-STATUS-MODULE-ACTIVATION-CONTROL	3	
		PGM-0190-400036-AV-400010-TABLE	3	X

PGM-0210	5	PGM-0210-400780-INDEX	2(3)	
		PGM-0210-MODULE-STATUS-MODULE-ACTIVATION-CONTROL	2(3)	
		PGM-0210-400036-AV-400010-TABLE	3	X
PGM-0220	2			
PGM-0230	2			
PGM-0310	2			
PGM-0320	5	PGM-0320-VAR-AUX	3	
		PGM-0320-006530-RCDS-006200-DTL	3	
		PGM-0320-400266-BOMBA-400260-BOMBA	2(3)	
		PGM-0320-400700-CT-400680-MSG	2(3)	X
END-OF-JOB	2			
PGM-START	5	PGM-START-006220-CFF-006200-DTL	2(3)	
		PGM-START-006215-PN-POS-1-006200-DTL	2(3)	
		PGM-START-006229-LOC-006200-DTL	2(3)	
		PGM-START-006230-AV-006200-DTL	2(3)	
		PGM-START-006285-RECUP-POR-006200-DTL	2(3)	
		PGM-START-006253-UN-006200-DTL	2(3)	
		PGM-START-006375-LAST-ACQ-PRICE-006200-DTL	2(3)	
		PGM-START-006246-BL-006200-DTL	2(3)	
		PGM-START-006250-NOMEN-006200-DTL	2(3)	
		PGM-START-006255-CAT-006200-DTL	2(3)	
		PGM-START-006260-OA-006200-DTL	2(3)	
		PGM-START-006263-APL-006200-DTL	2(3)	
		PGM-START-006265-TPR-006200-DTL	2(3)	
		PGM-START-006270-FRG-006200-DTL	2(3)	
		PGM-START-006280-TRG-006200-DTL	2(3)	
		PGM-START-006376-PROC-IN-REWORK-006200-DTL	2(3)	
		PGM-START-006377-COND-IN-REWORK-006200-DTL	2(3)	
		PGM-START-006380-SUPERADOR-006200-DTL	2(3)	
		PGM-START-006390-SUPERADO-006200-DTL	2(3)	
		PGM-START-400300-C-400280-9-REC	3	X

	PGM-START-400080-CFF-400050-PN-CFF	3	X
	PGM-START-450030-X-SPACE-001100-MASTER-0	3	X
	PGM-START-400083-PQ-400050-PN-CFF	3	X
	PGM-START-400085-AV-400050-PN-CFF	3	X
	PGM-START-450100-FED-MFG-CDE-001100-MASTER-0	3	X
	PGM-START-400800-D-400790-DATA-RESP	2(3)	
	PGM-START-400820-M-400790-DATA-RESP	2(3)	
	PGM-START-400840-A-400790-DATA-RESP	2(3)	
	PGM-START-400115-DAY-400110-DATE	2(3)	X
	PGM-START-400120-ME-400110-DATE	2(3)	X
	PGM-START-400130-AN-400110-DATE	2(3)	X
	PGM-START-400740-DATE	2(3)	X
	PGM-START-400100-POS-400090-RESPONSE	3	X
	PGM-START-400190-INDEX	3	X
	PGM-START-400185-SWT-400180-TEST	3	X
	PGM-START-400550-AV-400510-ID	3	X
	PGM-START-400036-AV-400010-TABLE	3	X
	PGM-START-400530-LOC-400510-ID	3	X
	PGM-START-400033-LOC-400010-TABLE	3	X
	PGM-START-400210-0-CT	3	X
	PGM-START-SWITCH-0130-PATH-CONTROL-SWITCHES	3	X
	PGM-START-400263-BOMBA-400260-BOMBA	3	X
	PGM-START-400266-BOMBA-400260-BOMBA	3	X
	PGM-START-450040-PART-NO-001100-MASTER-0	3	X
	PGM-START-400780-INDEX,	3	X
	PGM-START-006530-RCDS-006500-TRLR	3	X
	PGM-START-006400-ALTERNADO-006200-DTL	2(3)	
	PGM-START-006290-ESTOQUE-006200-DTL	2(3)	
	PGM-START-006300-EC-006200-DTL	2(3)	
	PGM-START-006310-OS-006200-DTL	2(3)	

		PGM-START-006320-REP-006200-DTL	2(3)	
		PGM-START-006330-AVG-PRICE-006200-DTL	2(3)	
		PGM-START-006350-A-006200-DTL	2(3)	
		PGM-START-006287-CON-006200-DTL	2(3)	
		PGM-START-006430-PRE-CALC-006200-DTL	2(3)	
		PGM-START-006440-NMAX-CALC-006200-DTL	2(3)	
		PGM-START-006450-CON-TOTAL-006200-DTL	2(3)	
		PGM-START-006470-MES-RECEB-006200-DTL	2(3)	
		PGM-START-006481-Q-P-ART-006200-DTL	2(3)	
		PGM-START-006482-Q-COMPRADA-006200-DTL	2(3)	
		PGM-START-006360-SHELF-006200-DTL	2(3)	
		PGM-START-006480-ANO-RECEB-006200-DTL	2(3)	
		PGM-START-MODULE-STATUS-MODULE-ACTIVATION-CONTROL	3	X
		PGM-START-VAR-AUX	3	X
		PGM-START-400700-CT-400680-MSG	3	X
		PGM-START-400070-PN-400050-PN-CFF	3	X
C1AD99T1	1			

5.3-1 Class and Functionality Analysis.

The legacy system uses one input file(SYS0) and one output file(SYS5). The input file has one record description 001100-MASTER-0 while the output file has two record descriptions 0062-DTL and 006500-TRLR. The Working Storage Section is composed of 28 records.

Each of the category 2 and category 3 subprograms from the C1AD99T1 system should have been converted to the object-oriented paradigm using the prototype. This

would have resulted in an object-oriented design with 185 classes and 185 methods. The main program should have also been converted to a class and method.

The Sigma 3 conversion did not work well. The example below (Figure 14) of subprograms PGM-0160-400700-CT-400680-MSG and PGM-0320-400700-CT-400680-MSG shows the problem that occurred.

<pre> procedure PGM-0160-400700-CT-400680-MSG (40070-PN-400050-PN-CFF, 450040-PART-NO- 001100-MASTER-0, 400033-LOC-400010-TABLE, 450030-X-SPACE- 001100-MASTER-0, 006530-RCDS-006500-TRLR, HEX-1, 400340-OP, 400700-CT-400680-MSG) begin LOCAL-9 := 400070-PN-400050-PN-CFF; LOCAL-8 := 006530-RCDS-006500-TRLR; if 450040-PART-NO-001100-MASTER-0 > LOCAL-9 (1) then PGM-0170-400070-PN-400050-PN-CFF (LOCAL-9, 450040-PART-NO-001100-MASTER-0, 450030-X-SPACE-001100-MASTER-0) else if 400033-LOC-400010-TABLE (1) = "VASP" then if 450040-PART-NO-001100-MASTER-0 = LOCAL-9 (1) then PGM-0170-400070-PN-400050-PN-CFF (LOCAL-9, 450040-PART-NO-001100- MASTER-0, 450030-X-SPACE-001100- MASTER-0) else endif else endif endif; if 450040-PART-NO-001100-MASTER-0 <= LOCAL-9 (1) then if 400033-LOC-400010-TABLE (1) /= "VASP" then if 450040-PART-NO-001100-MASTER-0 /= LOCAL-9 (1) then PGM-0320-400700-CT-400680-MSG (LOCAL-8, HEX-1, 400340-OP, 400700- CT- 400680-MSG); PGM-0320-006530-RCDS-006500-TRLR (LOCAL-8, HEX-1,400340-OP) else endif else endif else endif end </pre>	<pre> class CLASS-31 attributes 400700-CT-400680-MSG, 400340-OP, HEX-1, 006530-RCDS-006500-TRLR, 450030-X-SPACE- 001100-MASTER-0, 400033-LOC-400010-TABLE, 450040-PART-NO- 001100-MASTER-0,400070-PN-400050-PN-CFF method PGM-0160-400700-CT-400680-MSG (C-31) begin LOCAL-9 := GET-400070-PN-400050-PN-CFF (C-31); LOCAL-8 := GET-006530-RCDS-006500-TRLR (C-31); if GET-450040-PART-NO-001100-MASTER-0 (C-31) > LOCAL-9 (1) then PGM-0170-400070-PN-400050-PN-CFF (LOCAL-9, 450040-PART-NO-001100 MASTER-0,450030-X-SPACE-001100-MASTER-0) else if GET-400033-LOC-400010-TABLE (C-31, 1) = "VASP" then if GET-450040-PART-NO-001100-MASTER-0 (C-31) = LOCAL-9 (1) then PGM-0170-400070-PN-400050-PN-CFF (LOCAL-9, 450040-PART-NO- 001100-MASTER-0, 450030-X-SPACE-001100- MASTER-0) else endif else endif endif; if GET-450040-PART-NO-001100-MASTER-0 (C-31) <= LOCAL-9 (1) then if GET-400033-LOC-400010-TABLE (C-31, 1) /= "VASP" then if GET-450040-PART-NO-001100- MASTER-0 (C-31) /=LOCAL-9 (1) then PGM-0320-400700-CT-400680-MSG (LOCAL-8, GET- HEX-1 (C-31), GET- 400340-OP (C-31),GET-400700-CT- 400680-MSG (C-31)); PGM-0320- 006530-RCDS-006500-TRLR(LOCAL-8, GET-HEX-1 (C-31), GET-400340-OP (C-31)) else endif else endif else endif end superclass USER-OBJECT </pre>
---	---

<pre> procedure PGM-0320-400700-CT-400680-MSG (006530-RCDS-006500-TRLR, HEX-1, 400340- OP, 400700-CT-400680-MSG) begin LOCAL-6 := 006530-RCDS-006500-TRLR; LOCAL-6 := HEX-1 + 400340-OP; 400700-CT-400680-MSG := LOCAL-6; write (RCBU::STD-OUTPUT, 400700-CT-400680- MSG) end procedure PGM-0170-400070-PN-400050-PN-CFF (400070-PN-400050-PN-CFF, 450040-PART- NO-001100-MASTER-0, 450030-X-SPACE-001100-MASTER-0) begin if 450030-X-SPACE-001100-MASTER-0 = "T" then else 400070-PN-400050-PN-CFF (1) := 450040- PART-NO-001100-MASTER-0 endif end procedure PGM-0320-006530-RCDS-006500-TRLR (006530-RCDS-006500-TRLR, HEX-1, 400340-OP) begin 006530-RCDS-006500-TRLR := HEX-1 + 400340-OP end </pre>	<pre> class CLASS-15 attributes 400700-CT-400680-MSG, 400340-OP, HEX-1, 006530-RCDS-006500-TRLR method PGM-0320-400700-CT-400680-MSG (C-15) begin LOCAL-6 := GET-006530-RCDS-006500-TRLR (C-15); LOCAL-6 := GET-HEX-1 (C-15) + GET-400340-OP (C-15); SET-400700-CT-400680-MSG (C-15, LOCAL-6); write (RCBU::STD-OUTPUT, GET-400700-CT-400680- MSG (C-15)) end superclass USER-OBJECT class CLASS-8 attributes 450030-X-SPACE-001100-MASTER-0, 450040-PART-NO-001100-MASTER-0, 400070-PN- 400050-PN-CFF method PGM-0170-400070-PN-400050-PN-CFF (C-8) begin if GET-450030-X-SPACE-001100-MASTER-0 (C-8) = "T" then else SET-400070-PN-400050-PN-CFF (C-8, 1, GET-450040-PART-NO-001100- MASTER-0 (C-8)) endif end superclass USER-OBJECT class CLASS-17 attributes 400340-OP, HEX-1, 006530-RCDS-006500-TRLR method PGM-0320-006530-RCDS-006500-TRLR (C-17) begin SET-006530-RCDS-006500-TRLR (C-17, GET-HEX-1 (C-17) + GET-400340-OP (C-17)) end superclass USER-OBJECT </pre>
---	--

Figure 14 Sigma 3 Conversion Example

In the PGM-0160-400700-CT-400680-MSG procedure, the LOCAL-8 parameter is a PBOI case 3. Each of HEX-1, 400340-OP and 400700-CT-400680-MSG is a PBOI case 1. The parameter LOCAL-8, corresponding to 006530-RCDS-006500-TRLR, should have been converted from an attribute of class-15 to a parameter of a class-15 method. Nevertheless, that did not happen. The HEX-1, 400340-OP and 400700-CT-

400680-MSG remained attributes of class-15 but were not removed as attributes of class-31.

In Maj. Sward's dissertation about PBOI methodology [19], an important point was not described explicitly. When converting PBOI case 1 it is necessary to change an instance of the class C2 (the class corresponding to the called subprogram) to a parameter of the method of the class C1 (the class corresponding to the calling subprogram). It is necessary to put an instance of the class C2 (the class corresponding to the called subprogram) as a parameter of the method of the class C1 (the class corresponding to calling subprogram). While converting, the data remains an attribute of class C2 (the class corresponding to the called subprogram) and is removed as an attribute of C1 (the class corresponding to calling subprogram).

The classes class-15, class-8, class-17 and class-31 should be converted as shown below in Figures 15 , 16 , 17 and 18.

```
class CLASS-15 attributes
  400700-CT-400680-MSG, 400340-OP, HEX-1,
method PGM-0320-400700-CT-400680-MSG ( C-15 , 006530-RCDS-006500-TRLR )
begin
  LOCAL-6 := 006530-RCDS-006500-TRLR;
  LOCAL-6 := GET-HEX-1 ( C-15) + GET-400340-OP ( C-15);
  SET-400700-CT-400680-MSG ( C-15, LOCAL-6);
  write ( RCBU::STD-OUTPUT, GET-400700-CT-400680-MSG ( C-15))
end
superclass USER-OBJECT
```

Figure 15 - Sigma 3 Conversion Example (CLASS-15)

The problems were: (a) the attribute 006530-RCDS-006500-TRLR was neither removed as an attribute of the class-15 nor converted to a parameter of the class.

(b) the LOCAL-6 assignment should have been changed from the GET- message to the 006530-RCDS-006500-TRLR parameter.

```
class CLASS-8 attributes
  450030-X-SPACE-001100-MASTER-0,
  450040-PART-NO-001100-MASTER-0,
method PGM-0170-400070-PN-400050-PN-CFF ( C-8 , 400070-PN-400050-PN-CFF)
begin
  if GET-450030-X-SPACE-001100-MASTER-0 ( C-8) = "T"
  then
  else
    400070-PN-400050-PN-CFF(1) :=
    GET-450040-PART-NO-001100-MASTER-0 ( C-8)
  endif
end
superclass USER-OBJECT
```

Figure 16 - Sigma 3 Conversion Example(CLASS-8)

The problems were: (a) the 400070-PN-400050-PN-CFF attribute of the class-8 was neither removed nor converted to a parameter of the class-8.

(b) the SET-400070-PN-400050-PN-CFF message should have been changed to 400070-PN-400050-PN-CFF(1) := GET-450040-PART-NO-001100-MASTER-0 (C-8) assignment.

```

class CLASS-17 attributes
  400340-OP, HEX-1
method PGM-0320-006530-RCDS-006500-TRLR ( C-17 , 006530-RCDS-006500-
TRLR ) begin
  006530-RCDS-006500-TRLR := GET-HEX-1 ( C-17) + GET-400340-OP ( C-17)
end
superclass USER-OBJECT

```

Figure 17 - Sigma 3 Conversion Example(CLASS-17)

The problems were: (a) the 006530-RCDS-006500-TRLR attribute of the class-17 was neither removed nor converted to a parameter of the class-17.

(b) the SET-006530-RCDS-006500-TRLR message should have been changed to 006530-RCDS-006500-TRLR := GET-HEX-1 (C-17) + GET-400340-OP (C-17) assignment.

```

class CLASS-31 attributes
  006530-RCDS-006500-TRLR,
  400033-LOC-400010-TABLE,
  400070-PN-400050-PN-CFF
method PGM-0160-400700-CT-400680-MSG ( C-31 , C-15 , C-8) begin
  LOCAL-9 := GET-400070-PN-400050-PN-CFF ( C-31);
  LOCAL-8 := GET-006530-RCDS-006500-TRLR ( C-31);
  if GET-450040-PART-NO-001100-MASTER-0 ( C-8) > LOCAL-9 ( 1)
  then PGM-0170-400070-PN-400050-PN-CFF
    ( LOCAL-9,GET-450040-PART-NO-001100-MASTER-0(C-8),
    GET-450030-X-SPACE-001100-MASTER-0(C-8))
  else
    if GET-400033-LOC-400010-TABLE ( C-31, 1) = "VASP"
    then if GET-450040-PART-NO-001100-MASTER-0 ( C-8) = LOCAL-9 ( 1)
      then PGM-0170-400070-PN-400050-PN-CFF
        ( LOCAL-9, GET-450040-PART-NO-001100-MASTER-0(C-8),
        GET-450030-X-SPACE-001100-MASTER-0(C-8))
      else endif
    else endif
  endif:

```

```

if GET-450040-PART-NO-001100-MASTER-0 ( C-8) <= LOCAL-9 ( 1)
then f GET-400033-LOC-400010-TABLE ( C-31, 1) /= "VASP"
  then if GET-450040-PART-NO-001100-MASTER-0 ( C-8) /=
    LOCAL-9 ( 1)
    then PGM-0320-400700-CT-400680-MSG
      ( LOCAL-8, GET-HEX-1 ( C-15), GET-400340-OP ( C-15),
      GET-400700-CT-400680-MSG ( C-15));
      PGM-0320-006530-RCDS-006500-TRLR
      ( LOCAL-8, GET-HEX-1 ( C-17), GET-400340-OP ( C-17))
    else endif
  else endif
else endif
end
superclass USER-OBJECT

```

Figure 18 - Sigma 3 Conversion Example(CLASS-31)

The problems were: (a) the attributes HEX-1, 400340-OP, 400700-CT-400680-MSG, 450030-X-SPACE-001100-MASTER-0 and 450040-PART-NO-001100-MASTER-0 were not removed as attribute of the class-31.

(b) the GET- messages should have had its parameters changed to C-15 in the PGM-0320-400700-CT-400680-MSG message, C-17 in the PGM-0320-006530-RCDS-006500-TRLR and C-8 PGM-0170-400070-PN-400050-PN-CFF.

The next step was the transformation (Sigma 3 option) of the subprograms that call the subprogram PGM-0160-400700-CT-400680-MSG (class-31) into classes. This transformation also changed the classes that had already been built in the previous transformation (class-31 for example). These kind of changes cause further changes: attributes of a class become parameters of the corresponding class method. The new

parameters are instances of other classes whose methods are called by the first class method. This procedure causes the generation of overlapping classes or duplicate object instances. The overlapping classes and duplicate object instances are solved during the transformation of the main program into the SYSTEM-CLASS class.

A class overlaps another class when an instance of each is built using at least one common data item. Duplicate object instances are separate object instances that are built from the same class using the same data items.

In the previous example, the transformation of the PGM-0160-400700-CT-400680-MSG, PGM-0170-400070-PN-400050-PN-CFF, PGM-0320-400700-CT-400680-MSG and PGM-0320-006530-RCDS-006500-TRLR programs generated class-15 and class-17 overlapping classes. More overlapping classes should have been generated during the transformations of the subprograms until the system root was reached.

During the transformation of the main program, when the object instances are created before each message that invokes a method, the overlapping classes should merge but, they did not. This step should have created every object instance required for the entire object-oriented design.

Let's suppose that the PGM-0130 was the main program, this would have resulted in a class CLASS-SYSTEM as in Figure 19.

Class-15 and class-17 are overlapping classes and it is necessary to merge them into a new class and create a single new instance built from the new class. Then, any instance of an overlapping class (C-15 and C-17) should be replaced by an instance of the new class.

```

class CLASS-SYSTEM attributes
method PGM-0160( )
begin
  ...
  C-15:=CREATE-CLASS-15(400700-CT-400680-MSG , 400340-OP , HEX-1)
  C-8:=CREATE-CLASS-8(450030-X-SPACE-001100-MASTER-0 , 450040-PART-
    NO-001100-MASTER-0)
  C-17:=CREATE-CLASS-17(HEX-1 , 400340-OP)
  C-31:=CREATE-CLASS-31(006530-RCDS-006500-TRLR , 400340-LOC ,
    400070-PN-400050-PN-CFF)
  PGM-0160-400700-CT-400680-MSG(C-31 , C-15 , C-8)
  ...
end
superclass USER-OBJECT

```

Figure 19 - Initial Class-System

The overlapping classes are merged into a new class by union of the attributes and methods of the merged classes. It also creates a new method to create the new class.

Therefore, the new class (class-1517) and the CLASS-SYSTEM should have been built as shown in Figures 20 and 21.

```

class CLASS-SYSTEM attributes
method PGM-0160( )
begin
  ...
  C-8:=CREATE-CLASS-8(450030-X-SPACE-001100-MASTER-0 , 450040-PART-
    NO-001100-MASTER-0)
  C-1517:=CREAT-CLASS-1517(400700-CT-400680-MSG , HEX-1 , 400340-LOC)
  C-31:=CREATE-CLASS-31(006530-RCDS-006500-TRLR , 400340-LOC ,
    400070-PN-400050-PN-CFF)
  PGM-0160-400700-CT-400680-MSG(C-31 , C-1517 , C-8)
  ...
end
superclass USER-OBJECT

```

Figure 20 - Final Class-System

```

class CLASS-1517 attributes
400700-CT-400680-MSG , 400340-LOC , HEX-1
method CREATE-CLASS-1517(A-400700-CT-400680-MSG , A-400340-LOC , A-
HEX-1 ) : a CLASS-1517
begin
  INST-CLASS-1517:= new(CLASS-1517)
  SET-400700-CT-400680-MSG(INST-CLASS-1517 , A-400700-CT-400680-MSG)
  SET-400340-LOC (INST-CLASS-1517 , A-400340-LOC)
  SET- HEX-1 (INST-CLASS-1517 , A- HEX-1)
  CREATE-CLASS-1517:=INST-CLASS-1517
end
method PGM-0320-400700-CT-400680-MSG ( C-15 , 006530-RCDS-006500-TRLR )
begin
  LOCAL-6 := 006530-RCDS-006500-TRLR;
  LOCAL-6 := GET-HEX-1 ( C-15) + GET-400340-OP ( C-15);
  SET-400700-CT-400680-MSG ( C-15, LOCAL-6);
  write ( RCBU::STD-OUTPUT, GET-400700-CT-400680-MSG ( C-15))
end
method PGM-0320-006530-RCDS-006500-TRLR ( C-17 , 006530-RCDS-006500-
TRLR ) begin
  006530-RCDS-006500-TRLR := GET-HEX-1 ( C-17) + GET-400340-OP ( C-17)
end
superclass USER-OBJECT

```

Figure - 21 New Class Originated from Overlapping Classes

The sample transformation of PGM-0160-400700-CT-400680-MSG, PGM-0320-400700-CT-400680-MSG, PGM-0170-400070-PN-400050-PN-CFF and PGM-0320-006530-RCDS-006500-TRLR into the GOM shows that each remaining class in the object-oriented design, after the merging process, will not have attributes in common. Almost all the sliced subprograms in the C1AD99T1 system have many parameters in common. The origin of all data items is in the main program and the subprogram PGM-0100-READ is responsible for treating/computing all the input and output data items of the system. All these characteristics show that the PBOI methodology should have

created just one class for the input and output file with several methods corresponding to the subprograms that deal with the data items. The subprograms that do not process the input and output data items and do not call other subprograms use Working Storage data items. However, the other subprograms that process the input/output data item, use the same Working Storage data items. Therefore, these subprograms will generate overlapping classes too.

The “behavior” of the transformation of the C1AD99T1 system into the GOM showed that the object-oriented design will have just two classes, one for the main program (C1AD99T1) and another with all the data items in the system as attributes and all methods corresponding to the system subprograms.

The sliced subprograms were analyzed in order to address the following fact. The overall functionality of the imperative design was proven to be maintained after the translation of the system into the GIM and the transformation into the GOM. The sliced subprograms are results of the first phase of the transformation of the system into the GOM. And, the methods in a class are a copy of the corresponding sliced subprogram. As the sliced subprograms are built based on the output parameters produced in a subprogram, the statements that do not deal with them are not considered a component of the sliced subprogram. Therefore, a subprogram that has output statements using an in parameter will disappear from the system. This characteristic causes an inconsistent functionality of the object-oriented design with the legacy system.

An example of this lost functionality (Figure 22) is demonstrated with the PGM-0140 imperative subprogram.


```

procedure RU::PGM-0140
( RU::SWITCH-0130-PATH-CONTROL-SWITCHES,
  RU::400350-DATE-MSG, RU::400263-BOMBA-400260-BOMBA,
  RU::400266-BOMBA-400260-BOMBA,
  RU::400100-POS-400090-RESPONSE, RU::400033-LOC-400010-TABLE,
  RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
  RU::HEX-1, RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
  RU::400036-AV-400010-TABLE, RU::006530-RCDS-006500-TRLR,
  RU::400340-OP, RU::400700-CT-400680-MSG,
  RU::FILLER-CT-400680-MSG, RU::FILLER-40-400680-MSG,
  RU::VAR-AUX
) begin
RU::SWITCH-0130-PATH-CONTROL-SWITCHES := 160;
RU::PGM-0190
( RU::400033-LOC-400010-TABLE,
  RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
  RU::HEX-1, RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
  RU::400036-AV-400010-TABLE);
write ( STD-OUTPUT, RU::400350-DATE-MSG);
write ( STD-OUTPUT, "E F.FECHAR OU C.CONTINUAR");
if RU::400100-POS-400090-RESPONSE ( 1) = "F"
then RU::400263-BOMBA-400260-BOMBA := " ";
  RU::400266-BOMBA-400260-BOMBA := " ";
RU::PGM-0320
( RU::006530-RCDS-006500-TRLR, RU::HEX-1, RU::400340-OP,
  RU::400700-CT-400680-MSG, RU::FILLER-CT-400680-MSG,
  RU::FILLER-40-400680-MSG, RU::400263-BOMBA-400260-BOMBA,
  RU::400266-BOMBA-400260-BOMBA, RU::VAR-AUX)
else endif;
RU::PGM-0190
( RU::400033-LOC-400010-TABLE,
  RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
  RU::HEX-1, RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
  RU::400036-AV-400010-TABLE)
end

```

Figure 22 - Loss of Functionality (Slicing Problem)

The eliminated output statements showed 400350-DATE-MSG data item and asked for an operator intervention to continue the process or stop it. Therefore, as the

output statements did not remain in the object-oriented design, the resulting system would have had its functionality changed.

This demonstration showed that it is necessary to change the slicing process to keep the statements that do not deal with the output parameters.

Another problem that generated a loss of functionality was when a message to call a method could not be properly positioned within a class. Examples (Figure 23) of this were the messages within the class-20 to the class-2 and class-4 methods. The message to PGM-0210-400780-INDEX method would have been sent before the message to PGM-0210-400036-AV-400010-TABLE method, because the PGM-0210-400780-INDEX method set the 400780-INDEX data item value to the HEX-1 value and the PGM-0210-400036-AV-400010-TABLE uses the 400780-INDEX value. Therefore, if there was a statement following the LOCAL-1 := GET-400780-INDEX (C-4) assignment that used the LOCAL-1 data item, the value of the LOCAL-1 would be incorrect.

5.4 Summary.

This chapter has provided the results of the transformation of the Cobol legacy system into the GOM using the PBOI methodology. The PBOI prototype showed some flaws during the transformation of the C1AD99T1 system and was hard to execute. This transformation demonstrated that the PBOI methodology applied to Cobol legacy systems was not direct. The methodology could be applied to the small Cobol sample, yet showed the same problems with the conversion of the PBOI Case parameters. The C1AD99T1 system was not a giant or different from Cobol systems found in many organizations.

```

class CLASS-20 attributes
400036-AV-400010-TABLE, HEX-1, 400780-INDEX,
450040-PART-NO-001100-MASTER-0, 400033-LOC-400010-TABLE
method PGM-0190-400780-INDEX ( C-20 ) begin
  LOCAL-5 := GET-400033-LOC-400010-TABLE ( C-20);
  LOCAL-4 := GET-400036-AV-400010-TABLE ( C-20);
  LOCAL-5 ( 1 ) := GET-450040-PART-NO-001100-MASTER-0 ( C-20);
  PGM-0210-400036-AV-400010-TABLE
  ( 400780-INDEX, HEX-1, LOCAL-5, LOCAL-4);
  PGM-0210-400780-INDEX ( 400780-INDEX, HEX-1, LOCAL-5)
end
superclass USER-OBJECT

class CLASS-4 attributes
400036-AV-400010-TABLE, 400033-LOC-400010-TABLE, HEX-1,
400780-INDEX
method PGM-0210-400036-AV-400010-TABLE ( C-4 ) begin
  LOCAL-1 := GET-400780-INDEX ( C-4);
  if GET-400033-LOC-400010-TABLE ( C-4, 1) = "VASP"
  then LOCAL-1 := GET-HEX-1 ( C-4);
    PGM-0220 ( 400036-AV-400010-TABLE, LOCAL-1)
  else endif
end
superclass USER-OBJECT

class CLASS-2 attributes
400033-LOC-400010-TABLE, HEX-1, 400780-INDEX
method PGM-0210-400780-INDEX ( C-2 ) begin
  if GET-400033-LOC-400010-TABLE ( C-2, 1) = "VASP"
  then SET-400780-INDEX ( C-2, GET-HEX-1 ( C-2))
  else endif
end
superclass USER-OBJECT

```

Figure 23 - Loss of Functionality (Messages Placed Incorrectly)

Thus, the PBOI prototype was viable just for a small Cobol program that neither has many paragraphs nor produces many output parameters.

VI. Conclusions and Suggestions

6.1 Introduction.

The purpose of this research was to establish the feasibility of the PBOI methodology in relation to Cobol legacy systems. Three fundamental aspects were investigated: the GIM, the GOM and the PBOI prototype.

The initial phase of this research was to transform the Cobol legacy C1AD99T1 system into the GIM. As the Cobol language has many constructs whose structures are different from those of the GIM, it was necessary to develop a system to transform the Cobol constructs into those more similar to the GIM constructs. Then, a translation system was developed to translate the Cobol constructs into the GIM.

The second phase was to run the PBOI prototype. The aim was to extract the objects from the GIM legacy system that had been saved in a persistent object base file. However, the PBOI prototype was specific for the Fortran Ballistic Missile system and for an old version of Refine software. Therefore, the PBOI prototype was modified to deal with both the Cobol legacy system and the new version of Refine software.

The following sections present some conclusions about the PBOI methodology.

6.2 GIM conclusions

During the translation of the Cobol system into the GIM, some problems were encountered. Some restrictions imposed by the GIM had to be overcome because it is impossible for a Cobol system to exist with such restrictions. The restrictions were

described in chapter four. Even though Cobol is unique among imperative languages in many ways, the GIM had equivalents form most of them.

The restriction that the GIM does not model heterogeneous data structures is one that is impossible to satisfy because a Cobol program is focused on the design and implementation of data structures [21]. In her dissertation, Capt. Diná Moraes proposed a way to represent records within the GIM [24]. The record transformation/elimination increased the program length, because this transformation duplicates the statements whose operands are group items.

The transformation of the statements that had multiple assignments increased the number of lines of the program. The code was extended for each assignment in that statement. The transformation of the *perform* statement also increased the number of lines because when the performed paragraphs were before the stop run statement, the code within the paragraphs was duplicated.

Another aspect that has not been addressed in this research is the *redefines* clause in the Data Description Entry of the Cobol Data Division. The redefines clause allows the same storage area to be described by different data description entries. It is a characteristic that is widely used and found in a Cobol system and should be addressed.

The *redefines* clause hides a functional specification. Therefore, each time an operation is performed over a record, the redefined record experiences the same operation and vice-versa. One way to address this problem is to extend the Cobol code during the transformation of the legacy system into code that is very similar to the GIM. In a case where the two data description entries have the same characteristics of a data item, the

code is extended by writing the same operations using the redefined record (or the original record that is not explicitly used in the operation).

In the case that the two data description entries have different characteristics of a data item, a solution should be to construct a record with a sequence of bytes with the same length of the original data entry. Later, a function can be defined to map the redefined record to the sequence of bytes and from the sequence of bytes to a record. This should be a piece of the solution that deals with the statements that use data entries, and which are redefined. Future research should explore the changes required to deal with the redefines clause with different data description entries.

A way to include the record structure in the GIM should be developed after redefining the domain model and the grammar. This modification should be valuable because the object-oriented languages use record structures.

6.3 GOM conclusions

The absence of heterogeneous data structures should be addressed in the GOM as well. A way to represent heterogeneous data structures(records) within the GOM would be to add a gom-record subclass of gom-data-type. Figure 24 shows the gom-data-type class and the new subclass gom-record.

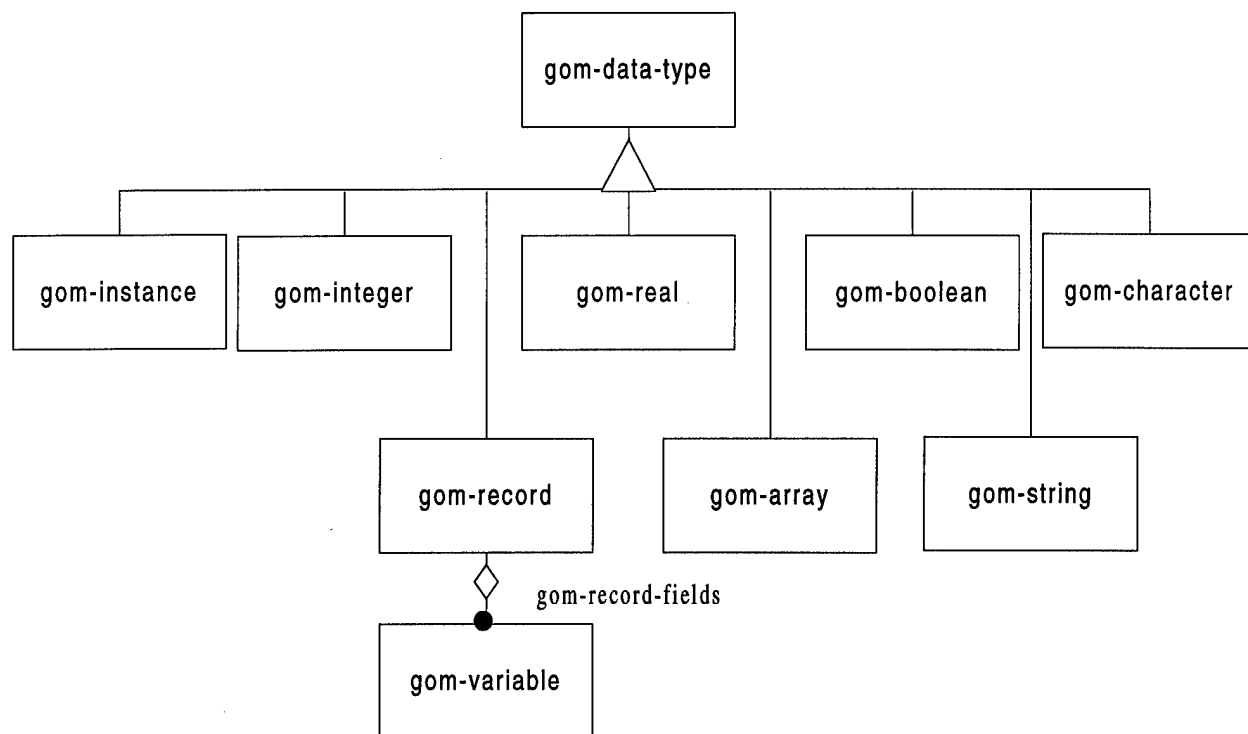


Figure 24 gom-record

6.4 Parameter-Based Object Identification Method Conclusion.

The PBOI method for identifying objects in imperative legacy code is based on the data items passed as parameters in imperative subprogram calls. This method is based on the thesis that object attributes manifest themselves as data items passed from subprogram to subprogram in the imperative paradigm[19].

After slicing and masking processes, as described in chapter II, the PBOI prototype starts the transformation of extracting objects into the GOM from the subprograms category 2 and 3 and the main program category 1.

The PBOI prototype is a powerful tool. It can automatically identify all the output parameters and construct the names of the program that are generated during the slicing process. But the entire process of slicing and masking is not automatic. It is

necessary for the operator/user to interact with the prototype to choose each sliced subprogram to mask. As the process of masking for each subprogram takes up to 20 minutes (depending on the quantity of output parameters produced in the subprogram), the whole process is slow taking a long time and needing a lot of interaction from the operator/user.

Slicing and masking again greatly expanded the size of the program because so many of the derived subprograms produced multiple, related outputs. The result was a large number of subprograms with many statements duplicated among several of them.

The prototype system is able to identify the main program in the PBOI methodology because the program has a specific name and is without parameters. So the imperative-symbol-table that is constructed during the transformation of the Cobol legacy system into the GIM, specifically when the parameters are translated, has its construction changed for the main program. Therefore, the imperative-symbol-table for the variables in the main program is built during the transformation of the statements in the main program.

When the source code scales up, specifically when there are many output parameters produced in a subprogram, the PBOI methodology is affected. It is affected because it provides many sliced programs and the PBOI prototype does not manage many output parameters and many subprograms well. Therefore, to transform the system into the GOM is more difficult for a Cobol system with many perform statements (calls to subprograms), because the structural complexity is increased.

This research has so far indicated that the approach of the PBOI methodology can be practically used in a small Cobol program that neither has many paragraphs nor

produces many output parameters. The real application of the approach will not be seen until a more robust and more automated PBOI system has been built.

6.5 Contributions.

This research has been completed successfully. The objectives defined for this work have been met.

This research makes the following major contributions:

1. Validation of the GIM using a Cobol legacy system;
2. Validation of the GOM with the records transformed into simple data type;
3. Demonstration that the PBOI prototype is impractical when applied to a system with several category 5 subprograms and many output parameters;
4. Demonstration that the Object-Oriented design is not consistent with the legacy code.

The analysis of the GIM, GOM and PBOI reveals a demonstration of the potentiality and flaws of the PBOI methodology as a generic reengineering tool for legacy systems. Also, my research provides substance for KBSE future research and for the PBOI methodology that Maj. Sward is applying in his work within the USAF.

The step of analyzing the extracted objects that are in the GOM was not accomplished. Consequently, it was impossible to verify their consistency with the original legacy system. Such verification was needed if the object-oriented design was to be shown to be functionally equivalent to the Cobol system. I was unable to evaluate the object-oriented design because of the PBOI prototype problems described in chapter V.

Despite the fact that the PBOI prototype was not capable of providing the object-oriented design of the legacy system, it was possible to conceive how the design might be.

Overall, the research demonstrated that while the PBOI methodology is a significant contribution in reengineering, it needs a better usage of elaborated types and a more powerful prototype to eliminate problems revealed during the transformation of the Cobol legacy system into the GOM.

Appendix A – Cobol Legacy System

000010 ID DIVISION.
C1AD10PC

000020* ESTA EH A REVISAO DE NUMERO 005
C1AD10PC

000030 PROGRAM-ID. C1AD99T1.
C01CMPPD

000040 AUTHOR. CONRAD G. WHITFIELD.
C01CMP

000050 INSTALLATION. DIRETORIA DE MATERIAL, FORCA AEREA BRASILEIRA,
C01CMP

000060 RIO DE JANEIRO, BRAZIL.
C01CMP

000070 DATE-WRITTEN. 02 OCT 1969.
C01CMP

000080 REMARKS.
C01CMP

000090 ***** HISTORIA DE MANUTENCAO DE PROGRAMA
*****C01CMP

000100	DATA	AUTORIDADE	DESCRICAO DE TROCO	POR
C01CMP				
000110	16-11-71		CONVERTIDO AO MESTRE	CGW
C01CMPOG				
000120			REV-70 E ANSI COBOL.	
C01CMPOG				
000130	16-01-84	SGT OSMAR	AUMENTEI 4 BYTES NOS ARQUIVOS	DCG
000140			DE ENTRADA, DEVIDO AOS MESTRES	
000150			ATUIAS TEREM 4 BYTES A MAIS; E	
000160			MOVI ESPACO ANTES DAS LEITURAS	
000170			FIM.	
000180	01-04-86	SGT EMILIA	TROCA DO PROCESSAMENTO DOS	DCG
000190			MESTRES EM FITA PARA DISCO	
000200			TAL COMO O ARQUIVO UNIFICADO.	
000210	04-10-88	SGT ROSANGELA	COM ALTERACAO NO REG. DO UNI-	DCG
000220			FICADO, COLOCANDO-SE O CAMPO	
000230			6450-CON-TOTAL.	
000240	26-10-88	SGT ROSANGELA	COM ALTERACAO NO REG. DO UNI-	DCG
000250			FICADO, COLOCANDO-SE O CAMPO	
000260			6460-DATA-RECEB.	
000270			E EXCLUINDO OS CAMPOS 006340-VALUE	
000280			006410-PRE E 006420-NMAX. ALTERANDO	
000290			DESTE MODO O TAMANHO DO REGISTRO CO-	
000300			MO TAMBEM O SEU NOME, QUE PASSOU A	
000310			SER C19N14PD.	
000320	25-08-92	SGT ROSANGELA	COM ALTERACAO NO REG. DO UNIFICADO	
000330			COLOCANDO-SE O CAMPO 6481-Q-P-ART	
000340			E 6482-Q-COMPRADA.	
000350				

000360 ENVIRONMENT DIVISION.
 000370 CONFIGURATION SECTION.
 000701 SPECIAL-NAMES.
 000702 console is console.
 000390 INPUT-OUTPUT SECTION.
 000410 FILE-CONTROL.
 000420 SKIP1
 000430 SELECT SYS0 ASSIGN TO SYS006-ARQ01
 000440 ORGANIZATION IS INDEXED
 000450 ACCESS MODE IS SEQUENTIAL
 000460 RECORD KEY IS 450040-PART-NO
 000470 FILE STATUS IS CHK-01.
 000480 SELECT SYS5 ASSIGN TO SYS011-UT-3350-AS
 000490 ORGANIZATION IS SEQUENTIAL
 000500 FILE STATUS IS CHK-UNIF.
 000520 DATA DIVISION.
 000530 FILE SECTION.
 000550 FD SYS0,
 000560 RECORD CONTAINS 448 TO 12488 CHARACTERS,
 000570 LABEL RECORDS ARE STANDARD.
 000590 01 001100-MASTER-0.
 000600 05 FILLER-1 PICTURE X(04).
 450030 05 450030-X-SPACE PICTURE X(01).
 450040 05 450040-PART-NO PICTURE X(18).
 450050 05 450050-AV-CODE PICTURE X(02).
 450060 05 450060-FED-STOCK-NO PICTURE X(15).
 450070 05 450070-NOMENCLATURE PICTURE X(14).
 450090 05 450090-REP-AT PICTURE X(03).
 450100 05 450100-FED-MFG-CDE PICTURE X(05).
 450110 05 450110-CATEGORY PICTURE X(01).
 450130 05 450130-LEAD-TIME PICTURE 9(02).
 450140 05 450140-SHELF-LIFE PICTURE 9(02).
 450160 05 450160-QUANT-PER-ART PICTURE X(04).
 450170 05 450170-HOURS PICTURE 9(04)V9.
 450210 05 450210-REWORK-FACT PICTURE 9(03).
 450230 05 450230-ACQ-PT PICTURE X(02).
 450240 05 FILLER-3 PICTURE X(13).
 450340 05 450340-REORDER-LEVEL PICTURE 9(04).
 450350 05 450350-MAX-STOCK PICTURE 9(05).
 450360 05 450360-TURN-AROUND PICTURE 9(03).
 450380 05 450380-ACCNT-IND PICTURE X(01).
 450390 05 450390-UNIT-OF-ISSUE PICTURE X(02).
 450410 05 450410-ON-ORD-QUANT PICTURE 9(05).
 450420 05 450420-REWORK-QUANT PICTURE 9(05).
 450430 05 450430-INV-BAL PICTURE 9(05).
 450440 05 450440-REM-BAL PICTURE 9(05).
 450450 05 450450-AVG-UNIT-PRICE PICTURE 999999V999.
 450470 05 450470-EXTENDED-VALUE PICTURE 9999999V999.
 450480 05 FILLER-4 PICTURE X(4).
 450520 05 450520-LAST-REC-DATE.
 450530 10 450530-LAST-REC-MO PICTURE 9(02).
 450540 10 450540-LAST-REC-YR PICTURE 9(02).
 450560 05 450560-LAST-PURCH-PRICE PICTURE 9(06)V999.
 450570 05 450570-REPAIRABLE-TOTAL PICTURE 9(04).
 450580 05 FILLER-5 PICTURE X(60).
 450750 05 450750-USAGE-TO-DATE PICTURE 9(06).
 450760 05 FILLER-6 PICTURE X(72).

450846	05	450846-CALC-PRE	PICTURE 9(05).
450847	05	450847-CALC-NMAX	PICTURE 9(05).
450848	05	450848-RENOV-HOLD	PICTURE 9(03).
450849	05	450849-CRIT-CTR	PICTURE 9(03).
450850	05	450850-ESTQ-DISP	PICTURE 9(05).
450851	05	450851-RENOV-CTR	PICTURE 9(03).
450852	05	450852-LAST-VEND	PICTURE X(05).
450860	05	450860-QUANT-SCRAPPED	PICTURE 9(06).
450870	05	450870-QUANT-PURCHASED	PICTURE 9(06).
450880	05	450880-EXPEND-TO-DATE	PICTURE 9(08)V99.
450890	05	450890-PROCESSED-IN-REWORK	PICTURE 9(06).
450900	05	450900-SCRAPPED-IN-REWORK	PICTURE 9(06).
450910	05	FILLER-7	PICTURE X(12).
450980	05	450980-REPLACING-PART-NUMBER	PICTURE X(18).
450990	05	450990-REPLACED-PART-NUMBER	PICTURE X(18).
451000	05	451000-ALTERNATE-PART-NUMBER	PICTURE X(18).
451020	05	451020-CON-MED	PICTURE 9(05)V9.
451030	05	451030-APPLICATION	PICTURE X(01).
451040	05	451040-INSTALL-TIME	PICTURE X(03).
451055	05	451055-PHYS-INV-SWT	PICTURE X(01).
000750	FD	SYS5,	
000755		LABEL RECORDS ARE STANDARD,	
006030		RECORD CONTAINS 222 CHARACTERS	
006050		DATA RECORDS ARE 006100-HDR, 006200-DTL, 006500-TRLR,	
006051		006600-LOC, 006700-TOT-RCD.	
006099			
006200	01	006200-DTL.	
006205	04	006205-ID.	
006210	05	006210-PN.	
006215	10	006215-PN-POS-1	PICTURE X(01).
006220	05	006230-AV	PICTURE X(05).
006224	05	006220-CFF	PICTURE X(05).
006227	05	006227-LOC.	
006229	10	006229-LOC	PICTURE X(02).
006240	05	006240-FSN.	
006246	10	006246-BL	PICTURE X(09).
006250	05	006250-NOMEN	PICTURE X(14).
006253	05	006253-UN	PICTURE X(02).
006255	05	006255-CAT	PICTURE X(01).
006260	05	006260-OA	PICTURE X(02).
006263	05	006263-APL	PICTURE X(01).
006265	05	006265-TPR	PICTURE 9(02).
006270	05	006270-FRG	PICTURE 9(03).
006275*05		006275-FRG-DEC REDEFINES 006270-FRG	PICTURE 9V99.
006280	05	006280-TRG	PICTURE 9(03).
006285	05	006285-RECUP-POR	PICTURE X(03).
006287	05	006287-CON	PICTURE 9(05)V9.
006290	05	006290-ESTOQUE	PICTURE 9(05).
006300	05	006300-EC	PICTURE 9(05).
006310	05	006310-OS	PICTURE 9(05).
006320	05	006320-REP	PICTURE 9(05).
006330	05	006330-AVG-PRICE	PICTURE 9(06)V999.
006340*05		006340-VALUE	PICTURE 9(07)V99.
006350	05	006350-A	PICTURE X(01).
006360	05	006360-SHELF	PICTURE 9(03).
006375	05	006375-LAST-ACQ-PRICE	PICTURE 9(06)V999.
006376	05	006376-PROC-IN-REWORK	PICTURE 9(06).

006377	05	006377-COND-IN-REWORK	PICTURE 9(06).
006380	05	006380-SUPERADOR	PICTURE X(18).
006390	05	006390-SUPERADO	PICTURE X(18).
006400	05	006400-ALTERNADO	PICTURE X(18).
006410	*05	006410-PRE	PICTURE 9(05).
006420	*05	006420-NMAX	PICTURE 9(05).
006430	05	006430-PRE-CALC	PICTURE 9(05).
006440	05	006440-NMAX-CALC	PICTURE 9(05).
006450	05	006450-CON-TOTAL	PICTURE 9(06).
006450	05	006460-DATA-RECEB.	
006450	10	006470-MES-RECEB	PICTURE 9(02).
006450	10	006480-ANO-RECEB	PICTURE 9(02).
006450	05	006481-Q-P-ART	PICTURE 9(04).
006450	05	006482-Q-COMPRADA	PICTURE 9(06).
006500	01	006500-TRLR.	
006510	05	006510-CTL	PICTURE X(32).
006520	05	006520-TRAILER-ID	PICTURE X(06).
006530	05	006530-RCDS	PICTURE 9(07).
000770		WORKING-STORAGE SECTION.	
001810	01	400680-MSG.	
001820	05	FILLER-CT	PICTURE X(32)
001830		VALUE IS '* REGISTROS MANDADOS PARA UNIFIC '.	
001840	05	FILLER-40	PICTURE X(06).
001840	05	400700-CT	PICTURE 9(07).
000790	01	CHK-01	PIC 9(02).
000840	01	CHK-UNIF	PIC 9(02).
000850	01	400010-TABLE VALUE IS ' '.	
000870	05	400030-ID OCCURS 5 TIMES.	
000880	10	400033-LOC	PICTURE X(04).
000890	10	400036-AV	PICTURE X(05).
000970	01	400090-RESPONSE.	
000980	05	400100-POS OCCURS 5 TIMES	PICTURE X(01).
000990	01	400110-DATE.	
001000	05	400115-DAY	PICTURE 9(02).
001010	05	400120-ME	PICTURE 9(02).
001020	05	400130-AN	PICTURE 9(02).
001030	01	400130-I VALUE SPACE	PICTURE X(01).
001050	01	400150-DATE.	
001000	05	400155-DAY	PICTURE 9(02).
001010	05	400160-ME	PICTURE 9(02).
001020	05	400170-AN	PICTURE 9(02).
001090	01	400180-TEST.	
001100	05	400185-SWT OCCURS 5 TIMES	PICTURE X(05).
001110	01	400190-INDEX USAGE IS COMPUTATIONAL	
001120		VALUE IS 1	PICTURE 9(01).
001140	01	400210-0-CT	PICTURE 9(07)
001150		USAGE IS COMPUTATIONAL, VALUE IS 0.	
001240	01	400260-BOMBA.	
001250	05	400263-BOMBA	PICTURE 9(01)
001260		VALUE IS ZERO.	
001270	05	400266-BOMBA	PICTURE 9(01)
001280		VALUE IS ZERO.	
001290	01	400280-9-REC .	
001310	05	400300-C OCCURS 5 TIMES	PICTURE X(01).
001360	01	400340-OP	PICTURE 9(07)
001370		USAGE IS COMPUTATIONAL, VALUE IS ZERO.	
001570	01	400480-UNITS.	

```

001580 05    FILLER-1                                PICTURE X(40)
001590      VALUE IS 'DISCO 1 **** DISCO 2 **** DISCO 3 '
001600 05    FILLER-2                                PICTURE X(30)
001610      VALUE IS '**** DISCO 4 **** DISCO 5 '
001620 01 400510-ID VALUE IS ' '.
001640 05    FILLER-0 OCCURS 5 TIMES.
001650 10    400530-LOC                                PICTURE X(04).
001660 10    FILLER-1                                PICTURE X(01).
001670 10    400550-AV                                PICTURE X(05).
001680 10    FILLER-2                                PICTURE X(05).
001860 01 400730-HOLD VALUE IS ZERO                 PICTURE 9(04).
001870 01 400740-DATE                               PICTURE 9(04).
001890 01 400790-DATA-RESP.
001900 05    400800-D                                PICTURE 9(02).
001920 05    400820-M                                PICTURE 9(02).
001940 05    400840-A                                PICTURE 9(02).
001970 01 HEX-0 USAGE IS COMPUTATIONAL VALUE IS 0, PICTURE 9(04).
001980 01 HEX-1 USAGE COMPUTATIONAL VALUE 1, PICTURE 9(04).
000910 01 400050-PN-CFF.
000920 05    400060-PN-CFF OCCURS 5 TIMES.
000930 10    400070-PN                                PICTURE X(18).
000940 10    400085-AV                                PICTURE X(05).
000950 10    400080-CFF                              PICTURE X(05).
000960 10    400083-PQ                                PICTURE X(04).
001380 01 400350-DATE-MSG                            PICTURE X(07).
002060 01 MODULE-ACTIVATION-CONTROL.
002090      02 MODULE-STATUS PIC X(30) VALUE ' '.
002100      02 PATH-CONTROL-VARIABLE PIC S9(4) COMP VALUE ZERO.
000775 01 VAR-AUX PIC X(01).
000780 01 END-OF-FILE PIC X(01).
001880 01 400780-INDEX USAGE COMPUTATIONAL PICTURE 9(04).
002110 01 PATH-CONTROL-SWITCHES.
002120      02 SWITCH-0130 PIC 9(4) COMP VALUE ZERO.
002130 PROCEDURE DIVISION.
002140 MAIN.
002150      PERFORM PGM-START THRU END-START.
002160      STOP RUN.
002170 END-OF-JOB.
002175      MOVE ' ' to VAR-AUX.
002175      DISPLAY 'STOP RUN' upon console.
002190 END-EOJ.
002200      EXIT.
002220 PGM-START.
002230*      *-----*
002240*      * PERFORMED BY MAIN. *
002250*      *-----*
002270      DISPLAY 'COM CCMP10. GERAR OS MESTRES REDUZIDOS P-300.'
002280      UPON CONSOLE.
002300
002320      OPEN
002330      OUTPUT
002340      SYS5.

```

```

002350     IF CHK-UNIF NOT = 00
002360         DISPLAY 'ERRO ABERTURA UNIF CKH = ' CHK-UNIF
002370         MOVE ' ' TO MODULE-STATUS
002380     else
002381
002390     MOVE ' ' TO 006200-DTL
002420     MOVE 10 TO 400790-DATA-RESP
002430     MOVE 400800-D TO 400115-DAY
002440     MOVE 400820-M TO 400120-ME
002450     MOVE 400840-A TO 400130-AN
002540     MULTIPLY 400130-AN BY 12 GIVING 400740-DATE
002550     ADD 400120-ME TO 400740-DATE
002560     MOVE '0020-600100' TO MODULE-STATUS.
002570     PERFORM PGM-0010 THRU 0010-END
002580     UNTIL MODULE-STATUS EQUAL ' '.
002590 END-START.
002600     EXIT.
002601
002620 PGM-0010.
002630* *-----*
002640* * PERFORMED BY START. *
002650* *-----*
002660     PERFORM PGM-0020 THRU 0020-END
002670     UNTIL MODULE-STATUS NOT EQUAL '0020-600100'.
002700     PERFORM PGM-0050 THRU 0050-END
002710     UNTIL MODULE-STATUS NOT EQUAL '0050-600300'.
002740     PERFORM PGM-0100-READ THRU 0100-END
002750     UNTIL MODULE-STATUS NOT EQUAL '0100-READ'.
002760     PERFORM PGM-0230 THRU 0230-END
002770     UNTIL MODULE-STATUS NOT EQUAL '0230-900073'.
002780     PERFORM PGM-0310 THRU 0310-END
002790     UNTIL MODULE-STATUS NOT EQUAL '0310-611330'.
002800 0010-END.
002810     EXIT.
002820
002830 PGM-0020.
002840* *-----*
002850* * PERFORMED BY PGM-0010. *
002860* *-----*
002870     MOVE ' ' TO MODULE-STATUS.
002880     DISPLAY ' DISCOS DE ENTRADA 01234'
002881
002890     MOVE ' ' TO 400090-RESPONSE.
002900     ACCEPT 400090-RESPONSE.
002910     MOVE HEX-0 TO 400190-INDEX.
002920     MOVE ' ' TO 400180-TEST.
002930     DISPLAY 'OS SEGUINTES DISCOS SERAO USADOS'.
002940     MOVE '0030-600140' TO MODULE-STATUS.
002950 0020-END.
002960     EXIT.
002970
003320

```



```

003780 PGM-0050.
003790* *-----*
003800* * PERFORMED BY PGM-0010. *
003810* *-----*
003820 MOVE ' ' TO MODULE-STATUS.
003830 ADD HEX-1 TO 400190-INDEX
003840 MOVE 400036-AV (400190-INDEX) TO 400550-AV
003850 (400190-INDEX).
003860 MOVE 400033-LOC (400190-INDEX) TO 400530-LOC
003870 (400190-INDEX).
003880 IF 400190-INDEX IS LESS THAN HEX-1
003890 MOVE '0050-600300' TO MODULE-STATUS
003900 OTHERWISE
003910 DISPLAY 400510-ID UPON CONSOLE
003950 MOVE '0060-610010' TO MODULE-STATUS.
003960 0050-END.
003970 EXIT.
003980* *** MAIN PROCESS ROUTINE ***
003990* ***.
004000
004860 PGM-0100-READ.
004870* *-----*
004880* * PERFORMED BY 0080-READ, PGM-0010, PGM-0090-READ. *
004890* *-----*
004900 MOVE ' ' TO MODULE-STATUS.
004910 READ SYS0
004920 IF END-OF-FILE = 'T'
004930 PERFORM PGM-0110 THRU 0110-END
004945 ELSE
004950 IF CHK-01 NOT = 00
004960 DISPLAY ' ERRO DE LEITURA SYS0 CHK = ' CHK-01
004970 CLOSE SYS0
004980 DISPLAY 'CLOSE SYS0'
004990 PERFORM END-OF-JOB THRU END-EOJ
005000 ADD HEX-1 TO 400210-0-CT
005010 PERFORM PGM-0130 THRU 0130-END.
005020 0100-END.
005030 EXIT.
005040* TO READ NEXT RECORD.
005050
005060 PGM-0110.
005070* *-----*
005080* * PERFORMED BY PGM-0100-READ. *
005090* *-----*
005100 IF 400300-C (1) IS EQUAL TO 'C'
005110 PERFORM PGM-0120 THRU 0120-END
005120 ELSE
005130 DISPLAY 'REGISTRO DE CONTROLE INEXISTENTE NO SYS0 DISC1'
005140 UPON CONSOLE
005170 PERFORM PGM-0120 THRU 0120-END.
005180 0110-END.
005190 EXIT.
005200

```

```

005300 PGM-0120.
005310* *-----*
005320* * PERFORMED BY PGM-0110. *
005330* *-----*
005340 DISPLAY 'SYS0 DISC1 FECHADO' UPON CONSOLE.
005350 MOVE HIGH-VALUES TO 400060-PN-CFF (1).
005360 CLOSE
005370 SYS0.
005380 DISPLAY 'FECHADO SYS0,ARQ01 CHK = ' CHK-01.
005490 0120-END.
005500 EXIT.
005510
005520 PGM-0130.
005530* *-----*
005540* * PERFORMED BY PGM-0100-READ. *
005550* *-----*
005560 IF SWITCH-0130 = 0160
005570 PERFORM PGM-0160 THRU 0160-END
005580 ELSE
005590 PERFORM PGM-0140 THRU 0140-END.
005600 0130-END.
005610 EXIT.
005630
005640 PGM-0140.
005650* *-----*
005660* * PERFORMED BY PGM-0130. *
005670* *-----*
005680 MOVE 0160 TO SWITCH-0130.
005750 PERFORM PGM-0190 THRU 0190-END
005790
005830 DISPLAY 400350-DATE-MSG UPON CONSOLE.
005840 DISPLAY 'E F..FECHAR OU C..CONTINUAR' UPON CONSOLE.
005860 IF 400100-POS(1) IS EQUAL TO 'F'
005870 MOVE ' ' TO 400260-BOMBA
005880 PERFORM PGM-0320 THRU 0320-END.
005890 PERFORM PGM-0190 THRU 0190-END.
005900 0140-END.
005910 EXIT.
005920
006080
006090 PGM-0160.
006100* *-----*
006110* * PERFORMED BY PGM-0130, 0150-900075. *
006120* *-----*
006130 IF 450040-PART-NO IS GREATER THAN 400070-PN (1)
006140 PERFORM PGM-0170 THRU 0170-END
006150 ELSE
006160* CHECK SEQUENCE OF MASTER AT 180.
006170 IF 400033-LOC (1) IS EQUAL TO 'VASP'
006180 IF 450040-PART-NO IS EQUAL TO 400070-PN (1)
006190 PERFORM PGM-0170 THRU 0170-END
006200 ELSE
006210 DISPLAY 'ERRO DA SEQUENCIA NO MESTRE SYS0 DISC1' UPON
006220 CONSOLE.
006225 IF 450040-PART-NO IS NOT GREATER THAN 400070-PN (1)
006227 IF 400033-LOC (1) IS NOT EQUAL TO 'VASP'
006228 IF 450040-PART-NO IS NOT EQUAL TO 400070-PN (1)

```

```

006230    DISPLAY 400070-PN (1) UPON CONSOLE
006240    DISPLAY 'ANTES' UPON CONSOLE
006250    DISPLAY 450040-PART-NO UPON CONSOLE
006260    MOVE ' ' TO 400260-BOMBA
006270    PERFORM PGM-0320 THRU 0320-END.
006280    0160-END.
006290    EXIT.
006300*                                     TO EOJ
ABNORM.
006310
006320 PGM-0170.
006330*    *-----*
006340*    *   PERFORMED BY PGM-0160.   *
006350*    *-----*
006360    IF 450030-X-SPACE IS EQUAL TO 'T'
006370        PERFORM PGM-0180 THRU 0180-END
006380    ELSE
006390        MOVE 450040-PART-NO TO 400070-PN (1)
006400        MOVE 450100-FED-MFG-CDE TO 400080-CFF (1)
006410        MOVE 400033-LOC (1) TO 400083-PQ (1)
006420        MOVE 400036-AV (1) TO 400085-AV (1).
006430    0170-END.
006440    EXIT.
006450*                                     TO EXIT.
006460
006470 PGM-0180.
006480*    *-----*
006490*    *   PERFORMED BY PGM-0170.   *
006500*    *-----*
006550    MOVE 'C' TO 400300-C (1).
006680    MOVE '0100-READ' TO MODULE-STATUS.
006690    0180-END.
006700    EXIT.
006710*                                     TO ABORT.
006720
006730 PGM-0190.
006740*    *-----*
006750*    *   PERFORMED BY PGM-0140.   *
006760*    *-----*
006770    MOVE 450040-PART-NO TO 400030-ID (1).
006780    PERFORM PGM-0210 THRU 0210-END.
006790    0190-END.
006800    EXIT.
006810
006920 PGM-0210.
006930*    *-----*
006940*    *   PERFORMED BY PGM-0190, 0200-900070.   *
006950*    *-----*
006960    IF 400033-LOC (1) IS EQUAL TO 'VASP'
006970        MOVE HEX-1 TO 400780-INDEX
006980        PERFORM PGM-0220 THRU 0220-END.
006990
007010    MOVE '0230-900073' TO MODULE-STATUS.
007020    0210-END.
007030    EXIT.
007040    SKIP2
007050*                                     **          ****          *

```

```

007060*           ** ANALYZE VASP LOCATION **
007070*           **           ****           **
007080
007090 PGM-0220.
007100* *-----*
007110* * PERFORMED BY PGM-0210. *
007120* *-----*
007130 IF 400036-AV (400780-INDEX) IS EQUAL TO 'S.TEC'
007140     MOVE 'VASPT' TO 400036-AV (400780-INDEX)
007150 ELSE
007160     MOVE 'VASP ' TO 400036-AV (400780-INDEX).
007170
007180 0220-END.
007190 EXIT.
007200
007210 PGM-0230.
007220* *-----*
007230* * PERFORMED BY PGM-0010, PGM-0090-READ. *
007240* *-----*
007250 MOVE ' ' TO MODULE-STATUS.
007260 MOVE '0100-READ' TO MODULE-STATUS.
007270 0230-END.
007280 EXIT.
007290*
007300*           ALTERED AT 900070 TO PROC VASP
007310*
007320*
007330*
007340*
007350*
007360*
007370*
007380*
007390*
007400*
007410*
007420*
007430*
007440*
007450*
007460*
007470*
007480*
007490*
007500*
007510*
007520*
007530*
007540*
007550*
007560*
007570*
007580*
007590*
007600*
007610*
007620*
007630*
007640*
007650*
007660*
007670*
007680*
007690*
007700*
007710*
007720*
007730*
007740*
007750*
007760*
007770*
007780*
007790*
007800*
007810*
007820*
007830*
007840*
007850*
007860*
007870*
007880*
007890*
007900*
007910*
007920*
007930*
007940*
007950*
007960*
007970*
007980*
007990*
008000*
008010*
008020*
008030*
008040*
008050*
008060*
008070*
008080*
008090*
008100*
008110*
008120*
008130*
008140*
008150*
008160*
008170*
008180*
008190*
008200*
008210*
008220*
008230*
008240*
008250*
008260*
008270*
008280*
008290*
008300*
008310*
008320*
008330*
008340*
008350*
008360*
008370*
008380*
008390*
008400*
008410*
008420*
008430*
008440*
008450*
008460*
008470*
008480*
008490*
008500*
008510*
008520*
008530*
008540*
008550*
008560*
008570*
008580*
008590*
008600*
008610*
008620*
008630*
008640*
008650*
008660*
008670*
008680*
008690*
008700*
008710*
008720*
008730*
008740*
008750*
008760*
008770*
008780*
008790*
008800*
008810*
008820*
008830*
008840*
008850*
008860*
008870*
008880*
008890*
008900*
008910*
008920*
008930*
008940*
008950*
008960*
008970*
008980*
008990*
009000*
009010*
009020*
009030*
009040*
009050*
009060*
009070*
009080*
009090*
009100*
009110*
009120*
009130*
009140*
009150*
009160*
009170*
009180*
009190*
009200*
009210*
009220*
009230*
009240*
009250*
009260*
009270*
009280*
009290*
009300*
009310*
009320*
009330*
009340*
009350*
009360*
009370*
009380*
009390*
009400*
009410*
009420*
009430*
009440*
009450*
009460*
009470*
009480*
009490*
009500*
009510*
009520*
009530*
009540*
009550*
009560*
009570*
009580*
009590*
009600*
009610*
009620*
009630*
009640*
009650*
009660*
009670*
009680*
009690*
009700*
009710*
009720*
009730*
009740*
009750*
009760*
009770*
009780*
009790*
009800*
009810*
009820*
009830*
009840*
009850*
009860*
009870*
009880*
009890*
009900*
009910*
009920*
009930*
009940*
009950*
009960*
009970*
009980*
009990*

```

```

009250     PERFORM END-OF-JOB THRU END-EOJ.
009260     0320-END.
009270     EXIT.
009280*           END OF ROUTINE TO READ SYS000-180.
009290*     SKIP3
009300*     NOTE
009310*           **           *****           **
009320*           **  ROTINA PARA PROCESSAR
009330*           **  ARQUIVO SYS001-281           **
009340*           **           *****           **
009340*     NOTE           *           *****           *
009350*           *  BUILD LOCACAO ID RECORD *
009360*           *           *****           *
009370

```

Appendix B – Legacy System Imperative Code

```
procedure RU::C1AD99T1 ( ) begin
  RU::PGM-START
  ( RU::006215-PN-POS-1-006200-DTL, RU::006230-AV-006200-DTL,
    RU::006220-CFF-006200-DTL, RU::006229-LOC-006200-DTL,
    RU::006246-BL-006200-DTL, RU::006250-NOMEN-006200-DTL,
    RU::006253-UN-006200-DTL, RU::006255-CAT-006200-DTL,
    RU::006260-OA-006200-DTL, RU::006263-APL-006200-DTL,
    RU::006265-TPR-006200-DTL, RU::006270-FRG-006200-DTL,
    RU::006280-TRG-006200-DTL, RU::006285-RECUP-POR-006200-DTL,
    RU::006287-CON-006200-DTL, RU::006290-ESTOQUE-006200-DTL,
    RU::006300-EC-006200-DTL, RU::006310-OS-006200-DTL,
    RU::006320-REP-006200-DTL, RU::006330-AVG-PRICE-006200-DTL,
    RU::006350-A-006200-DTL, RU::006360-SHELF-006200-DTL,
    RU::006375-LAST-ACQ-PRICE-006200-DTL,
    RU::006376-PROC-IN-REWORK-006200-DTL,
    RU::006377-COND-IN-REWORK-006200-DTL,
    RU::006380-SUPERADOR-006200-DTL,
    RU::006390-SUPERADO-006200-DTL,
    RU::006400-ALTERNADO-006200-DTL,
    RU::006430-PRE-CALC-006200-DTL,
    RU::006440-NMAX-CALC-006200-DTL,
    RU::006450-CON-TOTAL-006200-DTL,
    RU::006470-MES-RECEB-006200-DTL,
    RU::006480-ANO-RECEB-006200-DTL,
    RU::006481-Q-P-ARCE-006200-DTL,
    RU::006482-Q-COMPRADA-006200-DTL,
    RU::400800-D-400790-DATA-RESP,
    RU::400820-M-400790-DATA-RESP,
    RU::400840-A-400790-DATA-RESP, RU::400115-DAY-400110-DATE,
    RU::400120-ME-400110-DATE, RU::400130-AN-400110-DATE,
    RU::400740-DATE,
    RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL, RU::CHK-UNIF,
    RU::400100-POS-400090-RESPONSE, RU::400190-INDEX, RU::HEX-0,
    RU::400185-SWT-400180-TEST, RU::HEX-1,
    RU::400550-AV-400510-ID, RU::400036-AV-400010-TABLE,
    RU::400530-LOC-400510-ID, RU::400033-LOC-400010-TABLE,
    RU::FILLER-1-400510-ID, RU::FILLER-2-400510-ID,
    RU::400210-0-CT, RU::CHK-01, END-OF-FILE, RU::VAR-AUX,
    RU::SWITCH-0130-PATH-CONTROL-SWITCHES, RU::400350-DATE-MSG,
    RU::400263-BOMBA-400260-BOMBA,
    RU::400266-BOMBA-400260-BOMBA,
    RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
    RU::006530-RCDS-006500-TRLR, RU::400340-OP,
    RU::400700-CT-400680-MSG, RU::FILLER-CT-400680-MSG,
    RU::FILLER-40-400680-MSG, RU::400070-PN-400050-PN-CFF,
    RU::400080-CFF-400050-PN-CFF,
    RU::450100-FED-MFG-CDE-001100-MASTER-0,
    RU::400083-PQ-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
    RU::450030-X-SPACE-001100-MASTER-0,
    RU::400300-C-400280-9-REC)
end
```

```

procedure RU::END-OF-JOB ( RU::VAR-AUX ) begin
  RU::VAR-AUX := " "; write ( STD-OUTPUT, "STOP RUN" ) end

procedure RU::PGM-0010
  ( RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
    RU::400100-POS-400090-RESPONSE, RU::400190-INDEX, RU::HEX-0,
    RU::400185-SWT-400180-TEST, RU::HEX-1,
    RU::400550-AV-400510-ID, RU::400036-AV-400010-TABLE,
    RU::400530-LOC-400510-ID, RU::400033-LOC-400010-TABLE,
    RU::FILLER-1-400510-ID, RU::FILLER-2-400510-ID,
    RU::400210-0-CT, RU::CHK-01, END-OF-FILE, RU::VAR-AUX,
    RU::SWITCH-0130-PATH-CONTROL-SWITCHES, RU::400350-DATE-MSG,
    RU::400263-BOMBA-400260-BOMBA,
    RU::400266-BOMBA-400260-BOMBA,
    RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
    RU::006530-RCDS-006500-TRLR, RU::400340-OP,
    RU::400700-CT-400680-MSG, RU::FILLER-CT-400680-MSG,
    RU::FILLER-40-400680-MSG, RU::400070-PN-400050-PN-CFF,
    RU::400080-CFF-400050-PN-CFF,
    RU::450100-FED-MFG-CDE-001100-MASTER-0,
    RU::400083-PQ-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
    RU::450030-X-SPACE-001100-MASTER-0,
    RU::400300-C-400280-9-REC
  ) begin
  while
    not RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL /=
      "0020-600100"
    do begin
      RU::PGM-0020
      ( RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
        RU::400100-POS-400090-RESPONSE, RU::400190-INDEX, RU::HEX-0,
        RU::400185-SWT-400180-TEST)
      end;
    while
      not RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL /=
        "0050-600300"
      do begin
        RU::PGM-0050
        ( RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
          RU::400190-INDEX, RU::HEX-1, RU::400550-AV-400510-ID,
          RU::400036-AV-400010-TABLE, RU::400530-LOC-400510-ID,
          RU::400033-LOC-400010-TABLE, RU::FILLER-1-400510-ID,
          RU::FILLER-2-400510-ID)
        end;
      while
        not RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL /=
          "0100-READ"
        do begin
          RU::PGM-0100-READ
          ( RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
            RU::400210-0-CT, RU::HEX-1, RU::CHK-01, END-OF-FILE,
            RU::VAR-AUX, RU::SWITCH-0130-PATH-CONTROL-SWITCHES,
            RU::400350-DATE-MSG, RU::400263-BOMBA-400260-BOMBA,
            RU::400266-BOMBA-400260-BOMBA,
            RU::400100-POS-400090-RESPONSE, RU::400033-LOC-400010-TABLE,

```

```

RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
RU::400036-AV-400010-TABLE, RU::006530-RCDS-006500-TRLR,
RU::400340-OP, RU::400700-CT-400680-MSG,
RU::FILLER-CT-400680-MSG, RU::FILLER-40-400680-MSG,
RU::400070-PN-400050-PN-CFF, RU::400080-CFF-400050-PN-CFF,
RU::450100-FED-MFG-CDE-001100-MASTER-0,
RU::400083-PQ-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
RU::450030-X-SPACE-001100-MASTER-0,
RU::400300-C-400280-9-REC)
end;
while
not RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL /=
"0230-900073"
do begin
RU::PGM-0230 ( RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL)
end;
while
not RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL /=
"0310-611330"
do begin
RU::PGM-0310 ( RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL)
end
end
end

```

```

procedure RU::PGM-0020
( RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
RU::400100-POS-400090-RESPONSE, RU::400190-INDEX, RU::HEX-0,
RU::400185-SWT-400180-TEST
) begin
RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := " ";
write ( STD-OUTPUT, " DISCOS DE ENTRADA 01234");
RU::400100-POS-400090-RESPONSE := " ";
read ( FROM-CONSOLE, RU::400100-POS-400090-RESPONSE);
RU::400190-INDEX := RU::HEX-0;
RU::400185-SWT-400180-TEST := " ";
write ( STD-OUTPUT, "OS SEGUINTES DISCOS SERAO USADOS");
RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := "0030-600140"
end

```

```

procedure RU::PGM-0050
( RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
RU::400190-INDEX, RU::HEX-1, RU::400550-AV-400510-ID,
RU::400036-AV-400010-TABLE, RU::400530-LOC-400510-ID,
RU::400033-LOC-400010-TABLE, RU::FILLER-1-400510-ID,
RU::FILLER-2-400510-ID
) begin
RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := " ";
RU::400190-INDEX := RU::HEX-1 + RU::400190-INDEX;
RU::400550-AV-400510-ID ( RU::400190-INDEX) :=
RU::400036-AV-400010-TABLE ( RU::400190-INDEX);
RU::400530-LOC-400510-ID ( RU::400190-INDEX) :=
RU::400033-LOC-400010-TABLE ( RU::400190-INDEX);
if RU::400190-INDEX < RU::HEX-1

```



```

then RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL :=
    "0050-600300"
else
    write ( STD-OUTPUT, RU::400530-LOC-400510-ID);
    write ( STD-OUTPUT, RU::FILLER-1-400510-ID);
    write ( STD-OUTPUT, RU::400550-AV-400510-ID);
    write ( STD-OUTPUT, RU::FILLER-2-400510-ID);
    RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := "0060-610010"
endif
end

```

```

procedure RU::PGM-0100-READ
( RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
  RU::400210-0-CT, RU::HEX-1, RU::CHK-01, END-OF-FILE,
  RU::VAR-AUX, RU::SWITCH-0130-PATH-CONTROL-SWITCHES,
  RU::400350-DATE-MSG, RU::400263-BOMBA-400260-BOMBA,
  RU::400266-BOMBA-400260-BOMBA,
  RU::400100-POS-400090-RESPONSE, RU::400033-LOC-400010-TABLE,
  RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
  RU::400036-AV-400010-TABLE, RU::006530-RCDS-006500-TRLR,
  RU::400340-OP, RU::400700-CT-400680-MSG,
  RU::FILLER-CT-400680-MSG, RU::FILLER-40-400680-MSG,
  RU::400070-PN-400050-PN-CFF, RU::400080-CFF-400050-PN-CFF,
  RU::450100-FED-MFG-CDE-001100-MASTER-0,
  RU::400083-PQ-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
  RU::450030-X-SPACE-001100-MASTER-0,
  RU::400300-C-400280-9-REC
) begin
RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := " ";
read ( RU::SYS0,
  RU::FILLER-1-001100-MASTER-0,
  RU::450030-X-SPACE-001100-MASTER-0,
  RU::450040-PART-NO-001100-MASTER-0,
  RU::450050-AV-CODE-001100-MASTER-0,
  RU::450060-FED-STOCK-NO-001100-MASTER-0,
  RU::450070-NOMENCLATURE-001100-MASTER-0,
  RU::450090-REP-AT-001100-MASTER-0,
  RU::450100-FED-MFG-CDE-001100-MASTER-0,
  RU::450110-CATEGORY-001100-MASTER-0,
  RU::450130-LEAD-TIME-001100-MASTER-0,
  RU::450140-SHELF-LIFE-001100-MASTER-0,
  RU::450160-QUANT-PER-ART-001100-MASTER-0,
  RU::450170-HOURS-001100-MASTER-0,
  RU::450210-REWORK-FACT-001100-MASTER-0,
  RU::450230-ACQ-PT-001100-MASTER-0,
  RU::FILLER-3-001100-MASTER-0,
  RU::450340-REORDER-LEVEL-001100-MASTER-0,
  RU::450350-MAX-STOCK-001100-MASTER-0,
  RU::450360-TURN-AROUND-001100-MASTER-0,
  RU::450380-ACCNT-IND-001100-MASTER-0,
  RU::450390-UNIT-OF-ISSUE-001100-MASTER-0,
  RU::450410-ON-ORD-QUANT-001100-MASTER-0,
  RU::450420-REWORK-QUANT-001100-MASTER-0,
  RU::450430-INV-BAL-001100-MASTER-0,
  RU::450440-REM-BAL-001100-MASTER-0,

```

```

RU::450450-AVG-UNIT-PRICE-001100-MASTER-0,
RU::450470-EXTENDED-VALUE-001100-MASTER-0,
RU::FILLER-4-001100-MASTER-0,
RU::450530-LAST-REC-MO-001100-MASTER-0,
RU::450540-LAST-REC-YR-001100-MASTER-0,
RU::450560-LAST-PURCH-PRICE-001100-MASTER-0,
RU::450570-REPAIRABLE-TOTAL-001100-MASTER-0,
RU::FILLER-5-001100-MASTER-0,
RU::450750-USAGE-TO-DATE-001100-MASTER-0,
RU::FILLER-6-001100-MASTER-0,
RU::450846-CALC-PRE-001100-MASTER-0,
RU::450847-CALC-NMAX-001100-MASTER-0,
RU::450848-RENOV-HOLD-001100-MASTER-0,
RU::450849-CRIT-CTR-001100-MASTER-0,
RU::450850-ESTQ-DISP-001100-MASTER-0,
RU::450851-RENOV-CTR-001100-MASTER-0,
RU::450852-LAST-VEND-001100-MASTER-0,
RU::450860-QUANT-SCRAPPED-001100-MASTER-0,
RU::450870-QUANT-PURCHASED-001100-MASTER-0,
RU::450880-EXPEND-TO-DATE-001100-MASTER-0,
RU::450890-PROCESSED-IN-REWORK-001100-MASTER-0,
RU::450900-SCRAPPED-IN-REWORK-001100-MASTER-0,
RU::FILLER-7-001100-MASTER-0,
RU::450980-REPLACING-PART-NUMBER-001100-MASTER-0,
RU::450990-REPLACED-PART-NUMBER-001100-MASTER-0,
RU::451000-ALTERNATE-PART-NUMBER-001100-MASTER-0,
RU::451020-CON-MED-001100-MASTER-0,
RU::451030-APPLICATION-001100-MASTER-0,
RU::451040-INSTALL-TIME-001100-MASTER-0,
RU::451055-PHYS-INV-SWT-001100-MASTER-0);
if END-OF-FILE = "T"
then RU::PGM-0110
    ( RU::400300-C-400280-9-REC, RU::400070-PN-400050-PN-CFF,
      RU::400085-AV-400050-PN-CFF, RU::400080-CFF-400050-PN-CFF,
      RU::400083-PQ-400050-PN-CFF)
else
    if RU::CHK-01 /= 0
    then write ( STD-OUTPUT, " ERRO DE LEITURA SYS0 CHK = ");
        write ( STD-OUTPUT, "CLOSE SYS0");
        RU::END-OF-JOB ( RU::VAR-AUX);
        RU::400210-0-CT := RU::HEX-1 + RU::400210-0-CT;
        RU::PGM-0130
        (RU::SWITCH-0130-PATH-CONTROL-SWITCHES,
         RU::400350-DATE-MSG, RU::400263-BOMBA-400260-BOMBA,
         RU::400266-BOMBA-400260-BOMBA,
         RU::400100-POS-400090-RESPONSE, RU::400033-LOC-400010-TABLE,
         RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
         RU::HEX-1, RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
         RU::400036-AV-400010-TABLE, RU::006530-RCDS-006500-TRLR,
         RU::400340-OP, RU::400700-CT-400680-MSG,
         RU::FILLER-CT-400680-MSG, RU::FILLER-40-400680-MSG,
         RU::VAR-AUX, RU::400070-PN-400050-PN-CFF,
         RU::400080-CFF-400050-PN-CFF,
         RU::450100-FED-MFG-CDE-001100-MASTER-0,
         RU::400083-PQ-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
         RU::450030-X-SPACE-001100-MASTER-0,
         RU::400300-C-400280-9-REC)

```

```

    else endif
  endif
end

```

```

procedure RU::PGM-0110
  ( RU::400300-C-400280-9-REC, RU::400070-PN-400050-PN-CFF,
    RU::400085-AV-400050-PN-CFF, RU::400080-CFF-400050-PN-CFF,
    RU::400083-PQ-400050-PN-CFF
  ) begin
  if RU::400300-C-400280-9-REC ( 1 ) = "C"
  then RU::PGM-0120
    ( RU::400070-PN-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
      RU::400080-CFF-400050-PN-CFF, RU::400083-PQ-400050-PN-CFF)
  else
    write ( STD-OUTPUT,
      "REGISTRO DE CONTROLE INEXISTENTE NO SYS0 DISC1");
    RU::PGM-0120
    ( RU::400070-PN-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
      RU::400080-CFF-400050-PN-CFF, RU::400083-PQ-400050-PN-CFF)
  endif
end

```

```

procedure RU::PGM-0120
  ( RU::400070-PN-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
    RU::400080-CFF-400050-PN-CFF, RU::400083-PQ-400050-PN-CFF
  ) begin
  write ( STD-OUTPUT, "SYS0 DISC1 FECHADO");
  RU::400070-PN-400050-PN-CFF ( 1 ) := "9";
  RU::400085-AV-400050-PN-CFF ( 1 ) := "9";
  RU::400080-CFF-400050-PN-CFF ( 1 ) := "9";
  RU::400083-PQ-400050-PN-CFF ( 1 ) := "9";
  write ( STD-OUTPUT, "FECHADO SYS0,ARQ01 CHK = ")
end

```

```

procedure RU::PGM-0130
  ( RU::SWITCH-0130-PATH-CONTROL-SWITCHES,
    RU::400350-DATE-MSG, RU::400263-BOMBA-400260-BOMBA,
    RU::400266-BOMBA-400260-BOMBA,
    RU::400100-POS-400090-RESPONSE, RU::400033-LOC-400010-TABLE,
    RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
    RU::HEX-1, RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
    RU::400036-AV-400010-TABLE, RU::006530-RCDS-006500-TRLR,
    RU::400340-OP, RU::400700-CT-400680-MSG,
    RU::FILLER-CT-400680-MSG, RU::FILLER-40-400680-MSG,
    RU::VAR-AUX, RU::400070-PN-400050-PN-CFF,
    RU::400080-CFF-400050-PN-CFF,
    RU::450100-FED-MFG-CDE-001100-MASTER-0,
    RU::400083-PQ-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
    RU::450030-X-SPACE-001100-MASTER-0,
    RU::400300-C-400280-9-REC
  ) begin

```

```

if RU::SWITCH-0130-PATH-CONTROL-SWITCHES = 160
then RU::PGM-0160
  ( RU::400070-PN-400050-PN-CFF,
    RU::450040-PART-NO-001100-MASTER-0,
    RU::400033-LOC-400010-TABLE, RU::400263-BOMBA-400260-BOMBA,
    RU::400266-BOMBA-400260-BOMBA, RU::400080-CFF-400050-PN-CFF,
    RU::450100-FED-MFG-CDE-001100-MASTER-0,
    RU::400083-PQ-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
    RU::400036-AV-400010-TABLE,
    RU::450030-X-SPACE-001100-MASTER-0,
    RU::400300-C-400280-9-REC,
    RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
    RU::006530-RCDS-006500-TRLR, RU::HEX-1, RU::400340-OP,
    RU::400700-CT-400680-MSG, RU::FILLER-CT-400680-MSG,
    RU::FILLER-40-400680-MSG, RU::VAR-AUX)
else
RU::PGM-0140
  ( RU::SWITCH-0130-PATH-CONTROL-SWITCHES,
    RU::400350-DATE-MSG, RU::400263-BOMBA-400260-BOMBA,
    RU::400266-BOMBA-400260-BOMBA,
    RU::400100-POS-400090-RESPONSE, RU::400033-LOC-400010-TABLE,
    RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
    RU::HEX-1, RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
    RU::400036-AV-400010-TABLE, RU::006530-RCDS-006500-TRLR,
    RU::400340-OP, RU::400700-CT-400680-MSG,
    RU::FILLER-CT-400680-MSG, RU::FILLER-40-400680-MSG,
    RU::VAR-AUX)
endif
end

```

```

procedure RU::PGM-0140
  ( RU::SWITCH-0130-PATH-CONTROL-SWITCHES,
    RU::400350-DATE-MSG, RU::400263-BOMBA-400260-BOMBA,
    RU::400266-BOMBA-400260-BOMBA,
    RU::400100-POS-400090-RESPONSE, RU::400033-LOC-400010-TABLE,
    RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
    RU::HEX-1, RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
    RU::400036-AV-400010-TABLE, RU::006530-RCDS-006500-TRLR,
    RU::400340-OP, RU::400700-CT-400680-MSG,
    RU::FILLER-CT-400680-MSG, RU::FILLER-40-400680-MSG,
    RU::VAR-AUX
  ) begin
RU::SWITCH-0130-PATH-CONTROL-SWITCHES := 160;
RU::PGM-0190
  ( RU::400033-LOC-400010-TABLE,
    RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
    RU::HEX-1, RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
    RU::400036-AV-400010-TABLE);
write ( STD-OUTPUT, RU::400350-DATE-MSG);
write ( STD-OUTPUT, "E F.FECHAR OU C.CONTINUAR");
if RU::400100-POS-400090-RESPONSE ( 1) = "F"
then RU::400263-BOMBA-400260-BOMBA := " ";
    RU::400266-BOMBA-400260-BOMBA := " ";
RU::PGM-0320
  ( RU::006530-RCDS-006500-TRLR, RU::HEX-1, RU::400340-OP,

```

```

RU::400700-CT-400680-MSG, RU::FILLER-CT-400680-MSG,
RU::FILLER-40-400680-MSG, RU::400263-BOMBA-400260-BOMBA,
RU::400266-BOMBA-400260-BOMBA, RU::VAR-AUX)
else endif;
RU::PGM-0190
( RU::400033-LOC-400010-TABLE,
RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
RU::HEX-1, RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
RU::400036-AV-400010-TABLE)
end

procedure RU::PGM-0160
( RU::400070-PN-400050-PN-CFF,
RU::450040-PART-NO-001100-MASTER-0,
RU::400033-LOC-400010-TABLE, RU::400263-BOMBA-400260-BOMBA,
RU::400266-BOMBA-400260-BOMBA, RU::400080-CFF-400050-PN-CFF,
RU::450100-FED-MFG-CDE-001100-MASTER-0,
RU::400083-PQ-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
RU::400036-AV-400010-TABLE,
RU::450030-X-SPACE-001100-MASTER-0,
RU::400300-C-400280-9-REC,
RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
RU::006530-RCDS-006500-TRLR, RU::HEX-1, RU::400340-OP,
RU::400700-CT-400680-MSG, RU::FILLER-CT-400680-MSG,
RU::FILLER-40-400680-MSG, RU::VAR-AUX
) begin
if RU::450040-PART-NO-001100-MASTER-0
> RU::400070-PN-400050-PN-CFF ( 1)
then RU::PGM-0170
( RU::400070-PN-400050-PN-CFF,
RU::450040-PART-NO-001100-MASTER-0,
RU::400080-CFF-400050-PN-CFF,
RU::450100-FED-MFG-CDE-001100-MASTER-0,
RU::400083-PQ-400050-PN-CFF, RU::400033-LOC-400010-TABLE,
RU::400085-AV-400050-PN-CFF, RU::400036-AV-400010-TABLE,
RU::450030-X-SPACE-001100-MASTER-0,
RU::400300-C-400280-9-REC,
RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL)
else
if RU::400033-LOC-400010-TABLE ( 1) = "VASP"
then if RU::450040-PART-NO-001100-MASTER-0
= RU::400070-PN-400050-PN-CFF ( 1)
then RU::PGM-0170
(RU::400070-PN-400050-PN-CFF,
RU::450040-PART-NO-001100-MASTER-0,
RU::400080-CFF-400050-PN-CFF,
RU::450100-FED-MFG-CDE-001100-MASTER-0,
RU::400083-PQ-400050-PN-CFF, RU::400033-LOC-400010-TABLE,
RU::400085-AV-400050-PN-CFF, RU::400036-AV-400010-TABLE,
RU::450030-X-SPACE-001100-MASTER-0,
RU::400300-C-400280-9-REC,
RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL)
else
write ( STD-OUTPUT,
"ERRO DA SEQUENCIA NO MESTRE SYS0 DISC1")

```

```

        endif
    else endif
endif;
if RU::450040-PART-NO-001100-MASTER-0 <=
    RU::400070-PN-400050-PN-CFF ( 1)
then if RU::400033-LOC-400010-TABLE ( 1) /= "VASP"
    then if RU::450040-PART-NO-001100-MASTER-0 /=
        RU::400070-PN-400050-PN-CFF ( 1)
        then write ( STD-OUTPUT, RU::400070-PN-400050-PN-CFF ( 1));
        write ( STD-OUTPUT, "ANTES");
        write ( STD-OUTPUT, RU::450040-PART-NO-001100-MASTER-0);
        RU::400263-BOMBA-400260-BOMBA := " ";
        RU::400266-BOMBA-400260-BOMBA := " ";
        RU::PGM-0320
        (RU::006530-RCDS-006500-TRLR, RU::HEX-1, RU::400340-OP,
        RU::400700-CT-400680-MSG, RU::FILLER-CT-400680-MSG,
        RU::FILLER-40-400680-MSG, RU::400263-BOMBA-400260-BOMBA,
        RU::400266-BOMBA-400260-BOMBA, RU::VAR-AUX)
        else endif
    else endif
else endif
end

```

```

procedure RU::PGM-0170
( RU::400070-PN-400050-PN-CFF,
  RU::450040-PART-NO-001100-MASTER-0,
  RU::400080-CFF-400050-PN-CFF,
  RU::450100-FED-MFG-CDE-001100-MASTER-0,
  RU::400083-PQ-400050-PN-CFF, RU::400033-LOC-400010-TABLE,
  RU::400085-AV-400050-PN-CFF, RU::400036-AV-400010-TABLE,
  RU::450030-X-SPACE-001100-MASTER-0,
  RU::400300-C-400280-9-REC,
  RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL
) begin
if RU::450030-X-SPACE-001100-MASTER-0 = "T"
then RU::PGM-0180
    ( RU::400300-C-400280-9-REC,
      RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL)
else
    RU::400070-PN-400050-PN-CFF ( 1) :=
        RU::450040-PART-NO-001100-MASTER-0;
    RU::400080-CFF-400050-PN-CFF ( 1) :=
        RU::450100-FED-MFG-CDE-001100-MASTER-0;
    RU::400083-PQ-400050-PN-CFF ( 1) :=
        RU::400033-LOC-400010-TABLE ( 1);
    RU::400085-AV-400050-PN-CFF ( 1) :=
        RU::400036-AV-400010-TABLE ( 1)
endif
end

```

```

procedure RU::PGM-0180
( RU::400300-C-400280-9-REC,
  RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL
)

```

```

) begin
RU::400300-C-400280-9-REC ( 1) := "C";
RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := "0100-READ"
end

```

```

procedure RU::PGM-0190
( RU::400033-LOC-400010-TABLE,
  RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
  RU::HEX-1, RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
  RU::400036-AV-400010-TABLE
) begin
RU::400033-LOC-400010-TABLE ( 1) :=
  RU::450040-PART-NO-001100-MASTER-0;
RU::PGM-0210
( RU::400780-INDEX, RU::HEX-1, RU::400033-LOC-400010-TABLE,
  RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
  RU::400036-AV-400010-TABLE)
end

```

```

procedure RU::PGM-0210
( RU::400780-INDEX, RU::HEX-1, RU::400033-LOC-400010-TABLE,
  RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
  RU::400036-AV-400010-TABLE
) begin
if RU::400033-LOC-400010-TABLE ( 1) = "VASP"
then RU::400780-INDEX := RU::HEX-1;
  RU::PGM-0220 ( RU::400036-AV-400010-TABLE, RU::400780-INDEX)
else endif;
RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := "0230-900073"
end

```

```

procedure RU::PGM-0220
( RU::400036-AV-400010-TABLE, RU::400780-INDEX ) begin
if RU::400036-AV-400010-TABLE ( RU::400780-INDEX) = "S.TEC"
then RU::400036-AV-400010-TABLE ( RU::400780-INDEX) :=
  "VASPT"
else
  RU::400036-AV-400010-TABLE ( RU::400780-INDEX) := "VASP "
endif
end

```

```

procedure RU::PGM-0230
  ( RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL ) begin
  RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := " ";
  RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := "0100-READ"
end

procedure RU::PGM-0310
  ( RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL ) begin
  RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := " ";
  RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := "0060-610010"
end

procedure RU::PGM-START
  ( RU::006215-PN-POS-1-006200-DTL, RU::006230-AV-006200-DTL,
  RU::006220-CFF-006200-DTL, RU::006229-LOC-006200-DTL,
  RU::006246-BL-006200-DTL, RU::006250-NOMEN-006200-DTL,
  RU::006253-UN-006200-DTL, RU::006255-CAT-006200-DTL,
  RU::006260-OA-006200-DTL, RU::006263-APL-006200-DTL,
  RU::006265-TPR-006200-DTL, RU::006270-FRG-006200-DTL,
  RU::006280-TRG-006200-DTL, RU::006285-RECUP-POR-006200-DTL,
  RU::006287-CON-006200-DTL, RU::006290-ESTOQUE-006200-DTL,
  RU::006300-EC-006200-DTL, RU::006310-OS-006200-DTL,
  RU::006320-REP-006200-DTL, RU::006330-AVG-PRICE-006200-DTL,
  RU::006350-A-006200-DTL, RU::006360-SHELF-006200-DTL,
  RU::006375-LAST-ACQ-PRICE-006200-DTL,
  RU::006376-PROC-IN-REWORK-006200-DTL,
  RU::006377-COND-IN-REWORK-006200-DTL,
  RU::006380-SUPERADOR-006200-DTL,
  RU::006390-SUPERADO-006200-DTL,
  RU::006400-ALTERNADO-006200-DTL,
  RU::006430-PRE-CALC-006200-DTL,
  RU::006440-NMAX-CALC-006200-DTL,
  RU::006450-CON-TOTAL-006200-DTL,
  RU::006470-MES-RECEB-006200-DTL,
  RU::006480-ANO-RECEB-006200-DTL,
  RU::006481-Q-P-ART-006200-DTL,
  RU::006482-Q-COMPRADA-006200-DTL,
  RU::400800-D-400790-DATA-RESP,
  RU::400820-M-400790-DATA-RESP,
  RU::400840-A-400790-DATA-RESP, RU::400115-DAY-400110-DATE,
  RU::400120-ME-400110-DATE, RU::400130-AN-400110-DATE,
  RU::400740-DATE,
  RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL, RU::CHK-UNIF,
  RU::400100-POS-400090-RESPONSE, RU::400190-INDEX, RU::HEX-0,
  RU::400185-SWT-400180-TEST, RU::HEX-1,
  RU::400550-AV-400510-ID, RU::400036-AV-400010-TABLE,
  RU::400530-LOC-400510-ID, RU::400033-LOC-400010-TABLE,
  RU::FILLER-1-400510-ID, RU::FILLER-2-400510-ID,
  RU::400210-0-CT, RU::CHK-01, END-OF-FILE, RU::VAR-AUX,
  RU::SWITCH-0130-PATH-CONTROL-SWITCHES, RU::400350-DATE-MSG,
  RU::400263-BOMBA-400260-BOMBA,
  RU::400266-BOMBA-400260-BOMBA,
  RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,

```



```

RU::006530-RCDS-006500-TRLR, RU::400340-OP,
RU::400700-CT-400680-MSG, RU::FILLER-CT-400680-MSG,
RU::FILLER-40-400680-MSG, RU::400070-PN-400050-PN-CFF,
RU::400080-CFF-400050-PN-CFF,
RU::450100-FED-MFG-CDE-001100-MASTER-0,
RU::400083-PQ-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
RU::450030-X-SPACE-001100-MASTER-0,
RU::400300-C-400280-9-REC
) begin
write ( STD-OUTPUT,
"COM CCMP10. GERAR OS MESTRES REDUZIDOS P-300.");
if RU::CHK-UNIF /= 0
then write ( STD-OUTPUT, "ERRO ABERTURA UNIF CKH = ");
RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := " "
else
RU::006215-PN-POS-1-006200-DTL := " ";
RU::006230-AV-006200-DTL := " ";
RU::006220-CFF-006200-DTL := " ";
RU::006229-LOC-006200-DTL := " ";
RU::006246-BL-006200-DTL := " ";
RU::006250-NOMEN-006200-DTL := " ";
RU::006253-UN-006200-DTL := " ";
RU::006255-CAT-006200-DTL := " ";
RU::006260-OA-006200-DTL := " ";
RU::006263-APL-006200-DTL := " ";
RU::006265-TPR-006200-DTL := " ";
RU::006270-FRG-006200-DTL := " ";
RU::006280-TRG-006200-DTL := " ";
RU::006285-RECUP-POR-006200-DTL := " ";
RU::006287-CON-006200-DTL := " ";
RU::006290-ESTOQUE-006200-DTL := " ";
RU::006300-EC-006200-DTL := " ";
RU::006310-OS-006200-DTL := " ";
RU::006320-REP-006200-DTL := " ";
RU::006330-AVG-PRICE-006200-DTL := " ";
RU::006350-A-006200-DTL := " ";
RU::006360-SHELF-006200-DTL := " ";
RU::006375-LAST-ACQ-PRICE-006200-DTL := " ";
RU::006376-PROC-IN-REWORK-006200-DTL := " ";
RU::006377-COND-IN-REWORK-006200-DTL := " ";
RU::006380-SUPERADOR-006200-DTL := " ";
RU::006390-SUPERADO-006200-DTL := " ";
RU::006400-ALTERNADO-006200-DTL := " ";
RU::006430-PRE-CALC-006200-DTL := " ";
RU::006440-NMAX-CALC-006200-DTL := " ";
RU::006450-CON-TOTAL-006200-DTL := " ";
RU::006470-MES-RECEB-006200-DTL := " ";
RU::006480-ANO-RECEB-006200-DTL := " ";
RU::006481-Q-P-ART-006200-DTL := " ";
RU::006482-Q-COMPRADA-006200-DTL := " ";
RU::400800-D-400790-DATA-RESP := 10;
RU::400820-M-400790-DATA-RESP := 10;
RU::400840-A-400790-DATA-RESP := 10;
RU::400115-DAY-400110-DATE := RU::400800-D-400790-DATA-RESP;
RU::400120-ME-400110-DATE := RU::400820-M-400790-DATA-RESP;
RU::400130-AN-400110-DATE := RU::400840-A-400790-DATA-RESP;
RU::400740-DATE := RU::400130-AN-400110-DATE * 12;

```

```

RU::400740-DATE :=
RU::400120-ME-400110-DATE + RU::400740-DATE;
RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL := "0020-600100"
endif;
while not RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL = " "
do begin
RU::PGM-0010
( RU::MODULE-STATUS-MODULE-ACTIVATION-CONTROL,
RU::400100-POS-400090-RESPONSE, RU::400190-INDEX, RU::HEX-0,
RU::400185-SWT-400180-TEST, RU::HEX-1,
RU::400550-AV-400510-ID, RU::400036-AV-400010-TABLE,
RU::400530-LOC-400510-ID, RU::400033-LOC-400010-TABLE,
RU::FILLER-1-400510-ID, RU::FILLER-2-400510-ID,
RU::400210-0-CT, RU::CHK-01, END-OF-FILE, RU::VAR-AUX,
RU::SWITCH-0130-PATH-CONTROL-SWITCHES, RU::400350-DATE-MSG,
RU::400263-BOMBA-400260-BOMBA,
RU::400266-BOMBA-400260-BOMBA,
RU::450040-PART-NO-001100-MASTER-0, RU::400780-INDEX,
RU::006530-RCDS-006500-TRLR, RU::400340-OP,
RU::400700-CT-400680-MSG, RU::FILLER-CT-400680-MSG,
RU::FILLER-40-400680-MSG, RU::400070-PN-400050-PN-CFF,
RU::400080-CFF-400050-PN-CFF,
RU::450100-FED-MFG-CDE-001100-MASTER-0,
RU::400083-PQ-400050-PN-CFF, RU::400085-AV-400050-PN-CFF,
RU::450030-X-SPACE-001100-MASTER-0,
RU::400300-C-400280-9-REC)
end
end

```

Bibliography

1. Byrne, Eric J. "A Conceptual Foundation for Software Reengineering" Proceedings of the International Conference on Software Maintenance. 216-235. IEEE Computer Society Press, Nov 1992.
2. Bennett, Keith. "Legacy System: Coping with Success", IEEE Software (January 1995)
3. Liu, S. S. N. Wilde. "Identifying Objects in a Conventional Procedural Language: An Example of Data Design Recovery." Proceedings of the Conference on Software Maintenance. 266-271. Nov 1990.
4. Sneed, H. M. "Planning the Reengineering of Legacy Systems." IEEE, 24-34 (Jan 1995).
5. Korson, Jim and John D. McGregor. "Object-Oriented: A Unifying Paradigm," Communications of the ACM, 33(9):40-60(Sep 1990).
6. Yang, H. and Chu, W. C. and Sun Y.. "A Practical System of Cobol Program Reuse for Reengineering." IEEE , 45-57 (1997).
7. Yoshino, T. and Uehara, S. and Ookubo, T. and Suguta, S. and Hotta, Y. and Sonobe, M. "Reverse Engineering from Cobol to Narrative Specification." IEEE, 284-290 (1995).
8. Sneed, H. M. "Migration of Procedurally Oriented Cobol Programs in an Object-Oriented Architecture." IEEE, 105-111 (1992).
9. Fantechi, A. and Nesi, P. and Somma, E. " Object Oriented Analysis of Cobol." IEEE, 157-164 (1997).
10. Livadas, P. E. and Johnson, T. "A New Approach to Finding Objects in Programs." Software Maintenance : Research and Practive, Vol. 6, 249-260 (1994).

11. De Lucia, A. and Di Lucca, G. A. and Fasolino, A. R. and Guerra, P. and Petruzzelli, S. "Migrating Legacy Systems Towards Object-Oriented Platforms." IEEE, 122-129 (1997).
12. Zimmer, J. A. "Restructuring for Style". Software-Practice and Experience, 20(4):365-389 (1990).
13. Miller, K. W. and Morell, L. J. and Stevens, F. "Adding data Abstraction to Fortran Software." IEEE 5(6):50-58 (1988).
14. Dietrich, Jr. and Nackman, L. R. and Grace, F. "Saving a Legacy with Objects." OOPSLA Conference Proceedings, Special issue of SIGPLAN Notices, 24(10):77-83, 1989.
15. Jacobson, I. "Reengineering of old Systems to an Object-Oriented Architecture." OOPSLA Proceedings, 340-350 (1991).
16. Wegner, P. "Dimensions of Object-Based Language Design." Proceedings of OOPSLA, 168-182 (1987).
17. Cimitile, A. and Visaggio, G. "Software Salvaging and the Call Dominance Tree." J. of Systems and Software, vol. 28, no. 2 117-127 (February 1995).
18. Weiser, M. "Program Slicing." IEEE Trans. On Software Engineering, 352-357 (July 1984).
19. Sward, R. E. "Extracting Functionally Equivalent Object-Oriented Designs from Legacy Imperative Code". PhD. Dissertation, Air Force Institute of Technology (1997).
20. Horwitz, S., et al. "Interprocedural Slicing Using Dependence Graphs." Proceeding of the ACM SIGPLAN 88, Conference on Programming Language Design and Implementation. 35-46. (Jun 1998).
21. Hongji Yang, William C. Chu, and Young Sun. "A Practical System of COBOL Program Reuse for Reengineering," IEEE Eighth International Workshop on Technology and Engineering Practice incorporating Computer Aided Software Engineering, (1997).

22. Harmer, Terence J., McParland, Patrick J. and James M. Boyle. "Transformations to Restructure and Re-engineer COBOL Programs," Automated Software Engineering, (1998).
23. Leite, Julio Cesar , Sant'anna, Marcela and Prado, Antonio. "Porting Cobol Programs Using a Transformational Approach," Software Maintenance: Research and Practice , Vol. 9, 3-31 (1997).
24. Moraes, Diná Leite. "Transforming Cobol Legacy Software to a Generic Imperative Model" Master Dissertation, Air Force Institute of Technology (March - 1999).

VITA

Captain Sonia de Jesus Rodrigues was born on 25 May 1958 in Rio de Janeiro, Brazil. She graduated from Colégio João Alfredo in Rio de Janeiro in 1977. She entered undergraduate studies at Universidade Federal do Rio de Janeiro University in Rio de Janeiro, where she graduated with a Bachelor of Mathematics degree in Computer Science in June 1983.

Her first assignment was at Centro de Computação de Aeronáutica in March 1984.

Permanent Address: Rua Visconde de Itabaiana 160
Engenho Novo
Rio de Janeiro – RJ
Brazil
20780 – 180
fone: (055)(21) 281-8409

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 1999	3. REPORT TYPE AND DATES COVERED Master Dissertation	
4. TITLE AND SUBTITLE Cobol Reengineering Using the Parameter Based Object Identification (PBOI) Methodology			5. FUNDING NUMBERS	
6. AUTHOR(S) Sonia de Jesus Rodrigues, Captain, Brazilian Air Force				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology 2750 P St WPAFB, OH 45433-7765			8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/99J-02	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Roy F. Stratton AFRL/IFTD 525 Brooks Rd. Rome, NY 13441-4505			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This research focuses on how to reengineer Cobol legacy systems into object-oriented systems using Sward's Parameter Based Object Identification (PBOI) methodology. The method is based on relating categories of imperative subprograms to classes written in object-oriented language based on how parameters are handled and shared among them. The input language of PBOI is a canonical form called the generic imperative model (GIM), which is an abstract syntax tree (AST) representation of a simple imperative programming language. The output is another AST, the generic object model (GOM), a generic object oriented language. Conventional languages must be translated into the GIM to use PBOI. The first step in this research is to analyze and classify Cobol constructs. The second step is to develop Refine programs to perform the translation of Cobol programs into the GIM. The third step is to use the PBOI prototype system to transform the imperative model in the GIM into the GOM. The final step is to perform a validation of the objects extracted, analyze the system functionally, and evaluate the PBOI methodology in terms of the case study.				
14. SUBJECT TERMS Object-oriented model, reengineering, canonical forms, object identification.			15. NUMBER OF PAGES 149	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT	