Getting Objects Methods and Interactions by Extracting Business Rules from Legacy Systems

Omar EL BEGGAR, Brahim BOUSETTA and Taoufiq GADI
Faculty of Sciences and Techniques Settat
University Hassan 1
elbeggar_omar@yahoo.fr

DOI: 10.20470/jsi.v5i3.204

Abstract: The maintenance of legacy systems becomes over the years extremely complex and highly expensive due to the incessant changes of company activities and policies. In this case, a new or an improved system must replace the previous one. However, replacing those systems completely from scratch is also very expensive and it represents a huge risk. The optimal scenario is evolving those systems by profiting from the valuable knowledge embedded in them. This paper aims to present an approach for knowledge acquisition from existing legacy systems by extracting business rules from source code. In fact, the business rules are extracted and assigned next to the domain entities in order to generate objects methods and interactions in an object-oriented platform. Furthermore, the rules translation in natural language is given. The aim is advancing a solution for re-engineering legacy systems, minimize the cost of their modernization and keep very small the gap between the company business and the renovated systems.

Key words: Business rules, model, metamodel, model transformation, business rules extraction, slicing, sequence diagram, domain class diagram, legacy systems, reverse engineering

1. Introduction

The previous information systems ran on traditional mainframe environment, are usually written in COBOL and stored their data in files; they are large and complex and known as legacy systems. Those systems have been successively maintained and upgraded due to several causes including correction of anomalies, extensibility of requirements, business rules changes. Thereby, the software quality degreases and code becomes progressively obsolete. This problem is known as the software erosion phenomenon (Visaggio, 2001) and (Castillo, et al., 2011). In this case, a new or an improved system must replace the previous one. However, replacing these systems completely from scratch is very expensive, and represents a huge risk. Additionally, the software embeds a significant amount of business knowledge that would be lost if entirely replaced (Paradauskas, et al., 2006). To cope with the software erosion phenomenon, reengineering is the best solution for obtaining improved systems without discarding the existing systems. According to the authors (Biggerstaff, T., 1989), (Binkley, 1998) and (Chikofsky, et al., 1990), Reengineering is a discipline that aims to extract design information, functional specifications and, eventually, requirements from the source code, documents and any other available information that can be useful to maintain legacy systems or redevelop them. Reengineering preserves the legacy knowledge of the systems and makes it possible to maintain software easily at a low cost. This paper proposes a Model Driven Reengineering (MDR) which is based on a set of models that store the embedded business knowledge and a set of model transformations to reconstitute them in a new form and the subsequent implementation of this new form.

Business Rules are the main knowledge embedded in legacy systems which have been coded over years. Business rules are submitted to several changes due to the incessant development of the enterprises, the amendment of the legislation and many other causes discussed by authors (Bajec, et al., 2005) and (Filipowska, et al.,20011). When an update occurs in the business processes or rules the corresponding segments of the software must be changed. In course of time, both the software and the business rules become increasingly distant and isolated from each other. However, our MDRE approach presents a solution to keep the gap between the business rules and applications by means of a set of models such as the Business Rule Model which captures and specifies business rules and the LinkRule Model to assign rules to the domain objects. In order to adequately minimize misunderstanding between business people, and facilitate the storage and codification of business
rules by IT programmers, our proposal presents also a set of model transformation to translate those rules in Natural Language (NL) conforming to the Object Management Group (OMG) standard: Semantics Business Vocabularies and Rules (SBVR). Finally, a studying of business rules correlation is performed to carry out the objects interactions that will be represented in the Sequence Diagram of System's Internal Behavior (SDSIB). In this paper, we emphasize that business rules may not just serve as a mechanism for making applications flexible, but could also be used as a bridge that helps to keep the entire information system aligned with its business. As the main contribution, we propose a methodology and requirements for a tool support that enables to both business people and developers extracting the business rules hidden in code and proposing a methodology to provide a part of system dynamic behavior by means of generating methods and objects interactions.

The remainder of the paper is organized as follows: Section 2 gives a brief overview of our proposal and the related works. Section 3 proceeds with our business rules extraction approach. The Section 4 will be devoted for modeling business rules and the link with the reuse system domain. The following sections 5 and 6 will be consecrated to present the model transformation to respectively generate objects operations and translating business rules in NL. The section 7 will be dedicated to study the business rules correlation and the resulting objects interactions in SDSIB. An evaluation will be proposed in section 8. Finally, a conclusion and perspectives of this work are provided.

2. Proposal overview and related works

2.1 Proposal overview

Principally, our MDRE approach consists of generating the objects methods and interactions by extracting business rules from legacy systems. Our methodology presents four main phases: Business Rules Extraction, Modeling business rules, model transformation to find objects operations and rules translating in NL and finally studying business rules correlation to find objects interactions (see Figure1).

![Fig. 1: The different phases of our MDRE approach](image)

Extraction: In the first phase, we parse the source code and we extract the business rules by applying a slicing algorithm after detecting the relevant point and variables that constitute the slicing criterion. The slicing algorithm starts by constructing first a Program Dependence graph (PDG) of the parsed code and applying next a backward static slicing algorithm to extract the code segments related to business rules.
Modeling BR: the second phase of our MDRE approach consists first of modeling the resulting slices that represent the business rules extracted from code in the BR model and their assignment to the domain entities or classes using the RuleLink model.

Transformation: Once the business rules are stocked in models and assigned to the domain, we perform a model transformation to generate the objects operations or methods that will feed subsequently the Domain Class Diagram (DCD). Another transformation will be performed to translate the rules in NL by means of RuleSpeak notation conforming to the standard SBVR.

System interaction: the final phase focus on the business rules correlations to represent the objects interactions in the SDSIB and design the business logic of the original systems in a new form that may be used to generate code application in object-oriented platform.

2.2 Related works

In this subsection, we will present many works in this area of research interested in reverse engineering process, and the principal difference with our approach. Generally, a reverse engineering process aims to analyze a subject system for two goals (Chikofsky, et al., 1990):

- To identify the system’s components and their interrelationships
- To create representations of the system in another form or at higher level of abstraction

Previous research on reverse engineering made great achievements concerning the first reverse engineering goal but there is very little researches in creating representation of the system in another form especially at higher level of abstraction. Generally, the majority of researchers don’t integrate generally the meta-modeling and meta-models in their reverse engineering process as higher abstract representation of a system aspect and they don’t benefit of the MDE advantages. Even though, the metamodel as it is defined by OMG is a special kind of model that specifies the abstract syntax of a modeling language. It can be understood as the representation of the class of all models expressed in that language. According to the four layers of metamodels hierarchy (OMG, 2011), using a metamodel means working in higher level of abstraction. Therefore, as it was defined by (Chikofsky, et al., 1990), to attain the second goal of reverse engineering, it’s judicious to use meta-modeling and benefit automatically of the different layers of MDA. Otherwise, according to (Rugaber, et al., 1993) evolving legacy system without benefit of a higher level of representation of functionality and structure presents risks of quality. (Delgado, et al., 2010) state that the main goal of MDE is to reduce the sensitivity of primary software artifacts to the inevitable changes that affect a software system. Our MDRE approach belongs to this area of research that includes techniques and concepts of MDE which places the metamodels in the core of the different phases of the reverse engineering process.

In other side, there are many methodologies concerning Business Rules Extraction (BRE). The first methodology that have been proposed by several researchers is based on data mining and it consist of extracting business rules from database via several data mining algorithms like neural network, clustering and regression tree (Seltono, 2000), (Farquad, et al., 2008) and (Elalfi, et al., 2004). This first kind of BRE approaches aims to find the association rules or correlation between stored data in order to construct a business rule. For example, the data about how customers have behaved in the past will allow us to efficiently segment them in groups according to some criteria and consequently deduce the business rules.

However, the second kind of methodologies is based on slicing techniques that are interested in cutting source code segments which correspond to business rules. For example, the approach proposed by (Huang, et al., 1996-a) consists of a set of steps that identify firstly the domain variables and extract next the relevant code segments by Generalized Program Slicing (GPS) and finally present the segment of extracted code in formula tables. The (Harry M, et al., 1996) approach applied also slicing techniques but within a manually input of slicing criterion. However, both methodologies do not incorporate the modeling and metamodeling techniques to represent the extracted business rules in another form. Nevertheless, the approaches presented by (Castillo, et al., 2014) and (Zou, et al., 2004) include the MDE principles to extract knowledge from legacy systems. (Zou, et al., 2004) propose a framework based on a set of heuristic rules for extracting business processes through the static analysis of the source code. In other side, (Castillo, et al., 2014) use the standard Knowledge Discovery Model (KDM) as the metamodel to represent all the different legacy software artifacts involved in a legacy information system.
Finally, we can also talk about other techniques of extracting business rules from other available resources like getting rules by interviewing business experts or simply looking for them in procedure manuals. But, unfortunately those two resources are insufficient.

3. Business rules extraction

3.1 Business rules definition and taxonomy

In literature, there are many interpretations of business rule (BR). However a common-sense definition was proposed by the standard OMG SBVR (OMG, 2008); “BR is a rule that is under business jurisdiction. According to the Business Rules Group (BRG, 2000), BR is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business”. An example of a business rule is: “The age of an employee must not exceed to 62 years old” or “Only a privileged customer can receive a discount”. Many authors and IS development communities have attempted to define taxonomy of BR and propose a classification. Related to (Ross, et al., 1997), (Zimbrao G., et al. 2003) and (von Halle, 2002), the four main categories of BR are explained below:

- Structural or definitional rules: rules that define a business term (object, attribute, association end...) and the relationships (fact) among those terms. For example, “an article has an expiration date”;
- Mandatory constraints: also called integrity constraints rules, these rules typically constrain either the structure or the behavior of objects or processes. For example, “each price of product is greater than zero”.
- Derivation rules: define how information in one form may be transformed into another form. Derivation rules can be divided into two subcategories: inference rules and a computation rules such as “A customer’s annual order volume is always computed as the sum of order total” or an inference, like “A customer is considered potential if he has placed more than three orders over 100$”.
- Action Assertion rules: also known as (Condition-Action rules), they specify the action or activity that will be performed if the specified condition is met; for example “the supervisor must treat the customer command if the request delay of the customer is over 2 days”.

The structural rules can be used to identify the domain class relationships and mandatory constraints may used to define integrity constraints in database. Meanwhile, the derivation and action assertion rules allow getting the objects methods and interactions. We will focus in the rest of this paper on those last two kinds of rules.

3.2 Business rules extraction by program slicing

Generally, legacy systems handle and process a large amount of data and operations such as transaction processing, data access, running errors and screen handling. Due to the great size and complexity of legacy applications, it is often hard to separate the Business and Non-Business related code. The Business Rule Extraction (BRE) is the approach of isolating the code segments which are directly related to the business rules. The BRE can hence significantly reduce the cost and increase accuracy of knowledge extraction from legacy code. Thus, the next paragraph will be dedicated to present our BRE based on program slicing.

3.2.1 Program slicing overview

The concept of program slicing was originally introduced in 1979 by (Weiser, 1979; 1981; 1982). The slice as it is defined according to 14; “Given an imperative program, a slice is an executable program whose behavior must be identical to the specified subset of the original program's behavior”. Otherwise, a slice is composed of all statements that affect the value of a variable V at a point p in program. The pair point p and the variable V represent the criterion of slicing, noted C<p, V>. We can distinguish two main categories of slicing: static slice (backward/forward) using Program Dependence Graph (PDG) useful for program comprehension (De Lucia, A., et al., 1996) and (Harman, 2001), reverse engineering (Canfora, et al., 1994) and (Simpson, et al., 1993) or maintenance (Cimitile, et al., 1996). And the dynamic slice used for testing and debugging (Binkley, 1998) and (Lyle, et al., 1987). In our approach we have chosen a backward static slice based on PDG. This choice will be justified and explained with more details in the following subsections.
3.2.2 Program Dependence Graph

The Program Dependence Graph is a labeled, directed and rooted graph, denoted by PDG(P). Each directed edge will connect between any two nodes that represent statements in program P. The PDG can be represented as 4-tuple \((N_i, E_i, N_{entry}, N_{exit})\), where \(N_i\) is a set of nodes, \(E_i\) is a set of edges and \(N_{entry}\) and \(N_{exit}\) represent start and end nodes. Each node in CFG (P) has a set of definition variables and reference variables, respectively noted DEF\((N_i)\) and REF\((N_i)\). DEF\((N_i)\) defines variables defined (assigned) at the Node \(N_i\). And REF\((N_i)\) defines variables that reference other one at the Node \(N_i\). In other hand, the PDG edges symbolize either:

- Control Dependence (Dashed Lines): between a predicate and their controlled statements. Typically, in COBOL program a predicate is expressed either as an iterative statement such as \((\text{PERFORM UNTIL}, \text{PERFORM VARYING} \quad \text{and} \quad \text{PERFORM n times})\) or as a conditional statement such as \((\text{IF ELSE}, \text{EVALUATE WHEN})\).

- Data Dependence (Solid Lines): between statements that define a variable and those that may reference it. For Example the following COBOL statements ACCEPT X and MOVE X TO Y should be connected via a data dependence edge because the second statement \((\text{MOVE})\) defines a variable Y and it references in parallel the variable X which is already defined in the first one(ACCEPT).

We can express those kinds of dependences in PDG using the following definitions 1 and 2:

**Definition 1:** Let \(N_i\) and \(N_j\) two nodes in a PDG. There is a control dependence edge from \(N_i\) to \(N_j\); noted \(N_i \rightarrow cd N_j\). if \(N_i\) represents a control predicate and \(N_j\) represents a Statement of the program immediately nested within the predicate \(N_i\).

**Definition 2:** Let \(N_i\) and \(N_j\) two nodes in a PDG. There is a data dependence edge from \(N_i\) to \(N_j\); noted \(N_i \rightarrow dd N_j\). if the equation below is realized:

\[
\text{DEF}(N_i) \cap \text{REF}(N_j) \neq \emptyset \quad \text{and} \quad \nexists \quad k \quad / \quad i < k < j \quad \text{and} \quad \text{DEF}(N_k) \cap \text{REF}(N_j) \neq \emptyset
\]

Regarding the construction of a PDG which corresponds to a program P, it can be deduced from a CFG related to the same program P according to the definition 3:

**Definition 3:** For a given CFG \((N_i, E_i, N_{entry}, N_{exit})\), the program dependence graph PDG is a 3-tuple \((N_i, E_d, N_{entry})\) deduced from CFG by applying those rules:

\[
N_i(PDG) = N_i(CFG) \setminus \{N_{exit}\}
\]

We can express those kinds of dependences in PDG using the following definitions 1 and 2:

**Definition 1:** Let \(N_i\) and \(N_j\) two nodes in a PDG. There is a control dependence edge from \(N_i\) to \(N_j\); noted \(N_i \rightarrow cd N_j\). if \(N_i\) represents a control predicate and \(N_j\) represents a Statement of the program immediately nested within the predicate \(N_i\).

**Definition 2:** Let \(N_i\) and \(N_j\) two nodes in a PDG. There is a data dependence edge from \(N_i\) to \(N_j\); noted \(N_i \rightarrow dd N_j\). if the equation below is realized:

\[
\text{DEF}(N_i) \cap \text{REF}(N_j) \neq \emptyset \quad \text{and} \quad \nexists \quad k \quad / \quad i < k < j \quad \text{and} \quad \text{DEF}(N_k) \cap \text{REF}(N_j) \neq \emptyset
\]

3.2.3 Slicing algorithm

In this subsection, we will present the mainly steps to perform our slicing algorithm in order to identify the business rules hidden in legacy programs. Our proposal starts by a prior step that consists on organizing the unstructured legacy program through eliminating the jump statement and deleting the useless ones such as commentaries and blank lines. In the second step, we determine the slicing criterion necessary to execute the slicing algorithm by means of detecting the set of relevant points and variables in code. The third step is devoted to apply a backward static slicing algorithm which
uses a program dependence graph and reachability techniques to identify the program segments that represent the business rules.

The older versions of COBOL did not hold certain advanced features present in recent languages which could only be simulated by using GO TO statements; COBOL legacy systems are usually filled of GO TO statements. Those legacy systems are also built years ago and modified several times by multiple maintenance programmers. In turn, the code becomes highly unstructured and hard to extract knowledge from Applying slicing algorithms and the conventional PDG-based slicing algorithms will produce incorrect results in the presence of this unstructured code (Ball, 1993), (Ball et al., 1993) and (Choi et al., 1994).

Our BRE approach consists of normalizing also the unstructured source code by means of removing jump instructions, i.e. GO TO statements. Many algorithms are available to eliminate these jump statements and convert unstructured COBOL to a structured one such control-flow normalization proposed by the authors (Moonen, 1996), (Janssen et al., 1996; 1997) or transformations rules algorithms based on ASF+SDF Meta-Environment tool presented by (Sellink et al., 2001) and (Zaadnoordijk, 2001). Finally, the new code shall be a structured code executed in a sequential way without jumps. This prior step is required to apply the PDG-based slicing algorithm. The following subsections will be devoted to explain our approach for extracting business rules based on a complete automatic identification of slicing criterion. Thus, the first step in our slicing algorithm consists of identifying the first slicing criterion: the set of valuable statements in code that usually highlight the result performed by a business rules. Once we identify valuable statements, the second criterion is obtained by detecting relevant variables from code through a set of heuristic rules. Finally, the slicing program can be applied based on those criteria.

3.2.4 Automatic identification of slicing criterion

Generally after having done a program calculation, an action was performed to store or save the result in the data repository or otherwise an action was executed in order to display it to users either on screens or in printed reports. Once the statements that enforce those actions were detected in the source code, the other elements of the rule can be extracted automatically. Those statements that we call in our approach the Valuable Output Statement (VOS) are the key to perform our slicing method and extract the segment of code necessary to represent the business rule. Thereby, the start point p of the slice criterion will be the number of the detected VOS in program. Usually, a VOS statement can be display information on screen or on report; statement which saves or updates records in flat file or in database; statements that send variable to CICS or calls external program. Those statements may be classified into two kinds: Those related to records and those related to working or temporary variables. The first kind of VOS related to temporary variable represent:

- Display variable on screen
- Send variable to CICS
- Call result variable from external programs

The second kind of VOS related to records represent:

- Display a line on report
- Save record in storage support
- Update record in storage support

In a typical large business application, we can find a huge number of variables used in code, but only a subset of them is suitable for identifying business rules, i.e., those that represent the result of the business rule. We call these variables Business Relevant Variables (BRV), and we need a method to identify those BRV. The following subsection describes our method for automatic detection of BRV.

In other side, a slicing criterion of a program is a pair C<p, V> where p represent a position of statement in code and V is a set of variables. But, this is not sufficient for business rule extraction because it does not involve the nature of the business rule that will be extracted (derivation, or action assertion), and the general program slicing often produces too large slices sometimes fuzzy slices to be manipulated (Huang et al., 1996-a). Moreover, a business rule may cover only a particular portion of a program. Otherwise, without included others criteria in slicing program, the resulting slices may contain irrelevant or unnecessary code (Huang et al., 1996-a). Therefore, in order to improve our slicing program, we have first extended the PDG graph by specifying the types of their nodes. And we added another element to the slicing criterion: the depth of slicing that defines a limit in program segmentation as well as a set of heuristic rules to identify the BRV.
The extension of the PDG graph is performed by assigning a type to each node in this graph. A node can be iterative related to the statements (PERFORM VARYING, PERFORM UNTIL and PERFORM n TIMES), a control node attached to the COBOL statements (IF and EVALUATE), calculating node related to the statements that describe a calculation algorithm (ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPUTE), an assignment node which defines a (MOVE, ACCEPT and READ) statements or an action node which triggers a statement that call an external program in COBOL source code (CALL) an action node can also an action that enables CRUD operations in flat files or database.

In other side, the nature of the business rule is so important in identifying the business relevant variable necessary to slicing program. Usually, a business rule is centered around a result variable that depends on input variables. Formally, a business rule can be expressed as a program segment F that returns a result variable R by means of a set of input variables I1, I2...In, F (I1, I2...In) = R. the result variable is the business relevant variable that we must identify to perform our slicing program. Naturally, a result variable is a domain variable referenced in the node that corresponds to a VOS in the PDG graph. Furthermore, it should be defined in predecessor nodes. That's means; the BRV should be used as a defined variable in a calculating or assignment node. For example in the previous Figure 2, the variables MAX and TOT which are defined in the nodes 4, 6 and 7 in the PDG before the nodes of valuable statements 8 and 9 that reveal the result of a preceding calculation incorporated in those nodes. In order to identify the BRV, the following heuristic rules were being applied:

Heuristic1. A relevant variable is a domain variable that can be a based or a derived property in a domain entity.
Heuristic2. A relevant variable should be referenced in the valuable output statement at a point p in PDG graph.
Heuristic3. A relevant variable must be defined in a calculating node or an assignment node that assigns a literal value to the relevant variable.

Those heuristics can be consolidated in the following equation:

\[
BRV = \begin{cases} 
V \in REF(N_i)/ N_i \equiv VOS \\
\exists k / k < i \text{ and } 
\end{cases}
\]

After applying the heuristics to identify the slicing criteria, we can use a program backward static slicing based on PDG to retrieve the relevant code segment that corresponds to the hidden business rules in code.

### 3.2.5 Computing backward static slices

Before performing the algorithm, a preprocessing phase is required to normalize the unstructured code. Then, The algorithm may be applied to build firstly a control flow graph matrix by means of the function `constructCFG()` that parses the source code and extract the different nodes related to the code statements. Each node in the CFG is associated with a statement and it has a number that represent the position of the statement in program. Firstly, all CFG nodes are marked unvisited. The function allows also getting the defined and referenced variables for each node.

The second step in our algorithm consists of constructing the PDG based on the CFG already performed by the first function as well as the whole edges that connect the PDG nodes. This step is supported by the function `constructPDG()`. The third step of our algorithm is devoted to detect the slicing criteria via the function `getCriteria()` that identifies the valuable output statements and the relevant variables. Finally, the recursive function `getSlice()` which has as arguments: node n, statement position p and depth d. The function checks if the node n is not visited yet, the node n is marked visited and it is added to the resulting slice. Next, For each outgoing edge from n to an unvisited node v, the node v is passed as new argument to the recursive function `getSlice()`, the depth value is incremented and v is marked visited and added to the resulting slice too. Otherwise, if the depth value is equal to a given value the execution of the function is stopped and the resulting slice is returned.
ALGORITHM 1. Backward Static Slices Computation

**Input**: DEF and REF are Vector<Var>, DEP is Vector<Edge>, CFG is a matrix<Node,DEF,REF>, PDG is a matrix<Node,DEF,REF,DEP>, Slice is a set<Node>

**Function ConstructCFG() : CFG**

Debut

```
while Not end of File
  read Statement
  Node N=new Node(k,Statement);
  N.setVisited(false)
  St[]=StringTokenizer(Statement)
  for i varying from 0 to length of St
    check St[i] to feed vectors DEF and REF by corresponding variables and mark the type of nodes
  CFG.add(N,DEF,REF)
  k++
end while
```

Return CFG

**Function ConstructPDG(CFG : matrix<Node,DEF,REF>) : PDG**

Debut

```
for i varying from 1 to CFG size
  for j varying from i-1 to 0
    if CFG(i).Node is a Control or a loop Node
      Edge E=new Edge(j,'control dependence')
      DEP.add(E)
    elseif it exists a variable in CFG(i).REF equals to a variable in CFG(j).DEF
      Edge E=new Edge(j,'data dependence')
      DEP.add(E)
      break
  end for
end for
PDG.add(CFG(i),DEP)
end for
Return PDG
```

**Function getCriteria(PDG:matrix<CFG,DEP>): criterion<n,V>**

Debut

```
if N.getStatement() start with "WRITE" or "REWRITE" or "DISPLAY" or "SEND"
  then n = N.getNumber() and VOS=N.getStatement()
  foreach variable V in VOS.REF
    for i varying from n to 0
      check V belongs to PDG(i).Node.type='calculating' or 'assignment literal value' and V belongs to PDG(i).DEF
      Criterion.add(n,V)
    end for
  end for
end if
```

Return criterion

**Function getSlice(Node,n,V,depth): slice**

Debut

```
if (!N.isVisited())
  slice.add(N)
  N.setVisited(true)
end if
if PDG(N.getNumber()).DEF is not empty then depth++
  foreach Edge E in PDG(N.getNumber()).DEP
    if depth<3 and !PDG(E.getNumber()).Node.isVisited
      slice.add(N)
      N.setVisited(true)
      getSlice(PDG(E.getNumber()).Node,V,depth)
    end if
  end foreach
end if
Return slice
```

End Fuction
After performing the backward static slicing algorithm based on PDG graph, the computational business rules related to the different extracted slices are following: the rule 1 allows calculating the subtotal of each ordered item if the stocked quantity is greater than the ordered quantity. This business rule is extracted by means of the detected criterion C<95, IT-SUBTOTAL>.

The second rule is extracted using the criterion C<98, PR-QTY> and it consists of upgrading the quantity stocked of the ordered product even if the same condition in rule 1 is met. However, the rules 3 and 5 calculate respectively the total of the command and the total of its sales for a given customer via the criteria C<104, CD-TOTAL> and C<107, CU-TOTSALES>. While, the rule 4 is an inference rule extracted by means of the detected criteria C<114, CU-STATUS> which determines the status of the customer following its total sales. The computational rule 6 determines the discount to apply on the total amount of the command following the status of the customers. In other side, the rules 7 and 8 enables actions rules which involve respectively the UPDATE and CREATE that should be activated on the satisfaction of the condition: the stocked quantity of the ordered product should be greater than the ordered quantity.

4. Modeling business rules

In order to capture and specify in NL all business rules already extracted by means of PDG program slicing, A business rule metamodel is proposed to save those rules in a model that specifies their different elements. In fact, the slices or code fragments extracted in the previous phase will be parsed to build the business rule model. Others models are required in this step: the Domain Class Diagram (DCD) which will be used to achieve the representation of the business rules in NL and a link model to connect the business rules to the domain entities. Those last models aim to strike a balance between natural (to reflect a user language) and formal (to reflect a machine language). However, in this section we will give an overview for the metamodels concerning the BR, DCD and BR link and the model transformation performed to obtain the resulting methods and generating business rules in NL.

4.1 Business rules metamodel

Generally, a BR metamodel should have the following properties. First, it should have an exhaustive and mutual exclusive typology to capture all possible types of business rule. Second, it should have structured forms of expressions for implementation purposes. Third, it should include rule management elements according to (Wan-Kadir, et al., 2004). The proposed metamodel (see Figure 3) defines the meta-classes: Structural, Action Assertion, Constraint and Derivation that correspond to the taxonomy of the business rules already discussed in the previous section 3. Derivation rules are further divided into the meta-classes Computation and Inference. Idem for the action assertion rules, there are two types:

- Enabler meta-class which allows instancing the rules related to file records handling (Create, Update and Delete).
- Trigger meta-class which allows instancing rules that elicit an activity or a process to initiate.

Concerning the derivation rules, a set of business terms that may represent rule results or rule arguments as well as some values related to the vocabulary used in the rule definition.

Each business rule specifies the action that should be activated on the satisfaction of a certain condition. The meta-class Condition can be simple or composed of at least two other simple conditions connected using logical operators such as AND, OR, or NOT. The below metamodel provides a classification from the functional aspect of business rules. However, our enviable purpose is to represent business rule in NL. Thus, the meta-class Expression can be used to capture, store in a specific language the bodies of both conditions and actions used in the business rules.
4.2 Connecting decoupled business rules with domain objects

Business rules are closely attached to the data model, since the rules are about terms and facts. Generally, the data model of a business denotes all the information that an organization can retain about the business. (Ross, et al., 1997) affirms that a business rule should be anchored to some business entities. In other side, the authors (Wan-Kadir, et al., 2004) present an approach that aims relating the evolving business rules to the software design in order to control the business rules changes that may bring high impact on both the business processes and the software itself.

Furthermore, the designers apply the General Responsibilities Assignment Software Patterns (GRASP) to assign the operations to the responsible domain classes such as Creator, Expert, Controller, low coupling and high Cohesion to improve the software scalability and maintenance. Since our approach aims to convert the business rules to operations in object oriented systems. Thus, associating model with business is very appreciated at this level. Therefore, business rules should be anchored to the data model. This subsection will be consecrated to present the domain class model and the link model to connect the decoupled business rules to the domain.

4.2.1 Domain Classes Diagram

The class diagram is one of the leading diagrams in a UML modeling. It allows dissecting the system by showing its different classes; it provides a static view of the modeled system. In the software development process, the class diagram is also used to represent a domain model by identifying the system’s classes that participate in the execution of any system functionality. This model can be recovered also by applying a reverse engineering approach that intends to convert legacy programs into Object Oriented architectures (Jackobson, I, et al.,1991), (Jarzabek, S., et al., 1994), (Di Lucca, G.A, et al., 2000) and (EL BEGGAR, et al., 2013b) and. All those works present different methodologies to identify concepts and represent them in a domain model, and try to establish relationships among those concepts within and across models. In (EL BEGGAR, et al., 2013b), we have proposed a model driven reverse engineering approach to detect the domain elements and construct a normalized and integrated relational database from records description in code and information embedded in data files that store those records. Therefore, we are just content in this phase of the software process to determine the different classes without bothering to find their operations because we will find them next through the transformation of business rules.
In Figure 4, we present the DCD metamodel which is a reduced and simplified UML class diagram of (OMG, 2011) that respects the MOF specifications (OMG, 2006) and it will be extended with other elements like package and interface... Adding such elements will not affect significantly the transformation, few rules must be added. The DCD’s meta-model shows the classes with their owned attributes operations and relationships. The class which may inherit from other ones has several extremities or association ends. Each association end belongs to an association. An association is made between two classes or more and it has a name with at least two extremities. The different class attributes and operation parameters have also types which can be a simple data type or another classifier element in the domain.

4.2.2 Linked business rules to the domain

Our proposal aims transforming business rules to operations or methods that must be assigned to the responsible objects for their execution. However, the link or the relationship between business rules and domain objects is performed manually via the RuleLink metamodel presented below (See Figure 5). It describes the anchor between the business rule and the domain objects called ObjectExpert which is the principally responsible for the implementation of the business rule and hold in the matched method. A business rule link is also associated with zero or many domain elements such as a class, an association-end or an attribute that are necessary to define the resulting operation.

5. Getting objects methods from Business Rules

Once we have defined the metamodels required for models instantiation and validation. The remaining step to generate object methods according to the MDE principles is to define the mappings between those models in order to specify how each different model is going to be used during the model transformation. We used the ATLAS Transformation Language (ATL) (Allilaire, et al., 2006) and (ATL, 2006) that is a domain-specific language for specifying model-to-model (M2M) or model-to-text (M2T)
transformations. ATL is inspired by the OMG QVT requirements (OMG, 2005) and builds upon the OCL formalism. Regarding the definition of mappings rules, we have proposed the following method to develop our model transformation:

1. First, the mappings between models are defined using natural language.
2. Next, those mappings are structured by collecting them in a set of rules, expressed then in natural language.
3. Finally, the mapping rules are implemented using ATL language.

We will present then in table 1 the mainly mapping rules spelled in natural language and necessary to transform the BR to operations or methods that will be added to the domain classes. The Figure 6 highlights the target DCD operations obtained by applying those rules.

The model transformation performed in our proposal is a many-to-one and exogenous transformation which takes as source models: the BR model, the RuleLink model and DCD model. The DCD will be next fed by the resulting operations or methods by applying a set of matched rules expressed in ATL language that map the collected rules from legacy code to operations and assign them to the responsible objects in DCD model based on the anchors specified in the RuleLink manually between the business rules and the domain entities. In Figure 6, the computational rule 1 which calculate the item subtotal give place to the following operations `calculateIT-SUBTOTAL()` and `setIT-SUBTOTAL()` that assigns the value calculated previously by the first operation to the derived attribute `IT-SUBTOTAL` in the domain class `Item`. The Inference rule 2 is transformed to the setter operation `setCU-STATUS()` that affects the literal value "Potential" to the attribute `CU-STATUS` of the class `Customer` if the rule condition is met. In other side, the rule 3 is mapped to a create operation entitled `addItem()` that add the ordered item passed as parameter to a given `Command`.

**Table 1: The main mapping rules of the model transformation**

<table>
<thead>
<tr>
<th>Rule</th>
<th>From</th>
<th>TO</th>
<th>Mapping Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Inference (BR model)</td>
<td>Setters operation</td>
<td>Each Inference rule in the BR model gives place to the operation setter that assigns a value to the rule result. The operation has a parameter with the same type as BR result.</td>
</tr>
<tr>
<td>2</td>
<td>Computation (BR model)</td>
<td>Calculate and setter operations</td>
<td>The computation rule is mapped into two operation calculate that compute and returns the value of the rule result and a setter operation to assign its value to the derived attribute in the DCD. The calculate operation parameters are deduced from the business rule arguments.</td>
</tr>
<tr>
<td>3</td>
<td>Create (BR model)</td>
<td>Constructor and Add operations</td>
<td>The Action assertion create in the BR model will be transformed to a constructor charged to instantiate the subject object that will inserted into the container object in the DCD model via the operation add.</td>
</tr>
<tr>
<td>4</td>
<td>Delete (BR model)</td>
<td>Remove Operation</td>
<td>The action assertion delete becomes an operation remove which delete a domain object from its container object in the DCD model.</td>
</tr>
<tr>
<td>5</td>
<td>ObjectExpert (RL model)</td>
<td>Responsible object</td>
<td>The ObjectExpert is the object responsible that will contain the matched operations and it will be charged to their execution.</td>
</tr>
<tr>
<td>6</td>
<td>domainElement (RL model)</td>
<td>Operation parameters</td>
<td>The DomainElement specifies the domain elements in DCD such as an object, an attribute or an association-end that will be mapped into operations parameters.</td>
</tr>
<tr>
<td>7</td>
<td>DataType (DCD model)</td>
<td>Parameter and return types</td>
<td>To identify both the type of parameters and operation return, the meta-class <code>DataType</code> is used for constructing operation signatures.</td>
</tr>
</tbody>
</table>
6. Business rules formulation

The previous phase in our proposal is devoted to apply a backward static slicing program in order to extract the business rules as code segments and store them later in models within the original programming language (COBOL). While, this formulation of business rules is a lowest representation. It still appreciated by IT programmers that trust on code more than any other forms of business rule representation. Although, formulate the business rules in programming code is required; but make business rules more accessible and understandable by several system stakeholders such as business people is a judicious choice. In fact, the business rule needs a further formalization expressed in NL.

Formulation of business rules in NL is a higher level of representation that expresses those rules in accurate form. Consequently, the used business language must also be precise enough to adequately minimize misunderstanding between business people, and precise enough to facilitate their storage and codification by IT programmers. However, translating a piece of code related to a business rule from the original programming language to another make reuse of the existing code. Idem, the NL expressions may be translated to formal languages (such as Java, C#, C++) to makes them machine-processable. Recently, the OMG has published the semantics of business vocabulary and business rules (SBVR) specification (OMG, 2008) that defines the metamodel for documenting the semantics of business vocabulary, business facts and business rules. SBVR is targeted to capture business concepts and business rules in a language close enough to the ordinary language (such as RuleSpeak language and Structured English) to permit business experts to read them and keep them independent of any technical platform.

Firstly as preliminaries, we will present briefly the OMG’s standard SBVR and the chosen notation RuleSpeak that will be used to create templates for expressing the previous extracted business rules.
in NL. Next, a model to text transformation using ATL language will be performed to translate the business rules related to the defined templates expressed in RuleSpeak language.

6.1 Preliminaries

6.1.1 Semantics Business Vocabularies and Rules

In December 2007, the “Semantics of Business Vocabulary and Business Rules” (SBVR 1.0) was adopted by OMG as the specification for expressing business vocabularies and business rules in the language that is understandable by the business people without requiring any IT skills. Nevertheless, this specification can be used by IT people to perform transformations into information system models. The core idea of business rules formally supported by SBVR is the following: “Rules build on facts, and facts build on concepts as expressed by terms. Terms express business concepts; facts make assertions about these concepts; rules constrain and support these facts.” (OMG, 2008). However, the SBVR specification for expressing business rules is based on the following elements:

- Logical operations: SBVR supports distinct logical operations such as conjunction, disjunction, implication and negation.
- Quantification: The different quantifications used in SBVR consist of universal quantification, at least n quantification, at most n quantification, etc.
- Modal formulation: The modal formulations are also available in SBVR, e.g. “It is obligatory” or “It is necessary”.

The standard SBVR does not standardize any particular language or syntax for expressing vocabularies and rules. But, it still necessary for multi-lingual supports or notations such as Structured English, Object Rule Modeling (ORM) and RuleSpeak notation.

6.1.2 RuleSpeak notation

Business rules are usually informal statements; they require some more formal representation like logic language specification or graphical notation. RuleSpeak was one of the reference notations used in the creation of rules in accordance with the semantics specification of the SBVR. It is fully consistent with this standard and it represents a well-documented business rule notation. RuleSpeak is a natural language oriented business speech developed by Business Rule Solutions, LLC (BRS) and it has been used in large-scale projects since the late of the 1990s (OMG, 2008). However, the origin of RuleSpeak was started in 1996, when Ronald G. Ross developed a first version for English language before being adopted in different projects by BRS and having other translations in many language like Spanish, Dutch and Germany. The main guidelines of expressing a BR statement in RuleSpeak:

- Separate embedded calculations and conditions.
- BR statement must start with a singular subject (noun concept or term).
- The term must be attached to its context noun concept (item’s quantity).
- BR statement must have a Fact type (computed, considered, etc.).
- BR statement should include modality keywords (must, only if, always, sometimes…).
- BR statement should include quantifier keywords (Each, exactly one, a(n), etc.)
- Use RuleSpeak bullet list convention to express a combination of conditions. For example, a composed condition that concatenates simple conditions using the logical operator OR may be written as follows:

  A discount of 5% must be applied on any customer’s purchase only if any of the following conditions is true:
  — Today is Christmas.
  — The customer’s sales total is greater than 10000$.

Otherwise, a logical operator AND between simple conditions will be expressed as follows:

  A credit should be allowed to an individual only if all following conditions are true:
  — The individual’s age is less than 50 years.
  — The individual’s monthly salary must be greater than 750$.
6.2 Translating business rules to RuleSpeak notation

Before performing the translation, a preprocessing phase is required that focuses on the following steps: Firstly, the input expressions bodies concerning the actions and conditions of the extracted business rules within their original implementation code are read and broken into sentences. During sentence splitting, the margins of a sentence are identified and each sentence is separately stored. After sentence splitting, each sentence is processed to identify tokens and various parts of business rule speech. Once we get the preprocessing phase, the business rules will be mapped to a NL by using a model transformation technology. For model transformation of the primary business rules expressed in COBOL to NL, we need the following two requirements to perform the translation:

1. Selection of the appropriate RuleSpeak template.
2. Use of a set of mappings that can translate source elements expressed in original code to the equivalent elements used in RuleSpeak templates.

6.2.1 RuleSpeak templates

We have designed generic templates for the chosen business rules: inference rule, computation rule, and enabler rule. Following are the generic template written in RuleSpeak notation:

**Inference rule template:**
The `<domain property>` of the `<domain context class>` should be considered as `<value>` only if `<simple condition>` Or `<bullet list of a combination of conditions>`

**Computation rule template:**
The `<domain property>` of the `<domain context class>` should be computed as `<algorithm>` if only if `<simple condition>` Or `<bullet list of a combination of conditions>`

**Enabler rules template:**
The `<domain object>` should be added | removed | updated to | from | in its owner `<domain context class>` When `<simple condition>`

In all the above shown templates, elements written in brackets "[ ]" are optional. We get these elements from the original constructs of the business rules sentences expressed in COBOL code. The domain property and context class are terms extracted from the DCD model. The value assigned to the property in an inference rule is extracted from the BR model (see Figure 6). The simple condition is deduced from domain elements. However, the bullet list of a combination of conditions is used to translate the composed condition as it is explained in the subsection 6.1.2.

6.2.2 Model transformation from original code to RuleSpeak notation

The main purpose of this subsection is to simplify the understandability of the extracted business rules through using "business terms" instead of programming language constructs. The description of variables acronyms is required hence to formulate business rules in non-technical way. Note that each property in the DCD model presented above in subsection 6.2.1 is enriched by the element description that assigns a non-technical term to this property according to the enterprise policy and the adopted vocabularies.

In other side, as well as M2M transformation; the ATL language also enables developers to perform a M2T transformation by means of creating query that defines a transformation to a primitive data type especially a string type that is very useful to generate application source code or in our case expressing business rules in NL. Therefore, the next transformation in our proposal is a M2T transformation based on the BR model and DCD model by taking into consideration this time the description of the domain variables. And as target, it will be result a set of business rules expressed following the defined templates within RuleSpeak language. The ATL query defines a set of mappings that will used to translate business rule based code representation to RuleSpeak notation. Following is a brief overview of the used mappings from code representation to NL. The simple condition in BR expressed in COBOL can be concatenated using a relational operator. Following are the possible cases of relational operator and their translations:
Concerning the composed condition which is a combination of conditions using the logical operators.

Regarding the algorithms in computation rules, the COBOL representation uses a set of calculating verbs such as *ADD, SUBTRACT, MULTIPLY, DIVIDE*. Those verbs are translated into their corresponding nouns or subjects. However, the complex algorithm which includes embedded calculations will be separated into a bullet list of simple calculations. In other side, the statements which use a *COMPUTE* verb will be replaced by the previous verbs according to their arithmetic operators. For example, the following computation rule which calculates the customer’s purchase amount:

\[
\text{COMPUTE AMOUNT = AMOUNT} - \text{DISCOUNT} + \text{TAXES}
\]

The combined calculations will be split to the bullet list of simple calculations as follows:

\[
\begin{align*}
- & \text{SUBTRACT DISCOUNT FROM CD-AMOUNT} \\
- & \text{ADD TAXES TO CD-AMOUNT}
\end{align*}
\]

The enabler rules which allow performing operations in flat files repositories by means of the data access verbs such as *WRITE, DELETE, and REWRITE* will be translated into the following sentences:

\[
\begin{align*}
\text{write record} & \quad \text{An object is added to its context class} \\
\text{delete record} & \quad \text{An object is removed from its context class} \\
\text{rewrite record} & \quad \text{An object is updated in its context class}
\end{align*}
\]

Finally, for each term \( p1 \) which represent a based or a derived property in DCD must be mapped according to its description in the DCD model and it must be attached to its domain context class. Following is the translation of the business rules terms:
The Figure 7 shows an example of business rules translation in RuleSpeak notation according the whole templates explained below.

### COBOL representation

<table>
<thead>
<tr>
<th>Term</th>
<th>description term’s context class</th>
<th>description term of context class</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4-a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4-b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4-c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The resulting business rules expressed in RuleSpeak notation

- **R4-a:** The status of the customer should be considered as Potential only if the total sales of the customer is greater than 10000
- **R4-b:** The status of the customer should be considered as Medium only if all following conditions are true:
  - total sales of the customer is less than or equal to 10000
  - total sales of the customer is greater than or equal to 5000
- **R4-c:** The status of the customer should be considered as Normal only if the total sales of the customer is less than 5000

In this section we are concerned with one of our ultimate goals which is the study of the impact of business rules correlation on objects interactions making. In fact, the first subsection will be dedicated to present the SDSIB model and its metamodel. Next, in the second subsection the different possible types of correlation will be given in order to generate finally the objects interactions in the SDSIB model.

7. **Business rules Correlation and SDSIB interactions**

Business rules must give a mechanism to reuse systems and participate in the construction of recent programs. Usually, a group of business rules constraints defines and governs a business process related to the domain objects that participate to this process. When the status of an object changes or an interaction is performed between objects, a need is felt for applying related business rules. Therefore, compute dependencies between business rules will be useful to find objects interactions and construct the SDSIB. Otherwise, the decoupled rules cannot trace the impact of change in the business model through business rules, redundant rules will be appear and processes cannot incorporate rules efficiently.

In this section we are concerned with one of our ultimate goals which is the study of the impact of business rules correlation on objects interactions making. In fact, the first subsection will be dedicated to present the SDSIB model and its metamodel. Next, in the second subsection the different possible types of correlation will be given in order to generate finally the objects interactions in the SDSIB model.
7.1 Sequence Diagram of System Internal Behavior

7.1.1 SDSIB overview

Generally, in software development process there are two kinds of sequence diagrams that can be modeled during process life cycle: The Sequence Diagram of System External Behavior (SDSEB) in analysis phase and the Sequence Diagram of System Internal Behavior SDSIB in design phase. While, The (SDSEB) is a model at higher level of abstraction which shows only interactions between actors and system considered as unique entity without focusing on internal system objects interactions. The SDSIB is a model at lower level of abstraction which shows detailed interactions between the domain objects and the exchange of messages between them.

7.1.2 SDSIB metamodel

Figure 8 shows the proposed target metamodel of SDSIB. Each operation in SDSIB concerns an incoming message trigged by an actor that may be principal or secondary. The operation may have parameters and return a value and it is associated to the objects: source object that represents the object that will invoke the operation and target object that will contain and release the operation. The source and target objects should correspond to one of the following domain classes: view, controller or model. Each operation belongs to an interaction operand that can be placed in combined fragment (Loop, Alt, Opt...). Note, a combined fragment may be also nested in another Combined fragment.

Fig.8: SDSIB metamodel

7.2 Resulting objects interactions based on Business rules dependences

The motivation for computing correlations between business rules is to study the objects interaction among rules correlations and designing the SDSIB. While, business rules taxonomy classifies business rules into certain basic types; correlation between business rules determines their interrelationship and interaction between objects. Rules are not isolated from each other. They are related to some other rules. This correlation or relationship between business rules could be based on their common condition or predicate or could be correlated through their result. Rules may also be attached to their common object model. Finding correlation between business rules also helps in
combining the individual pieces of knowledge embedded in the rules into comprehensive knowledge of the rule base (Veerendra Kumar Rai, et al., 2004).

7.2.1 Business rules referred domain object

A basic grouping of business rules according to their referred domain object is a basic and natural kind of dependence between rules. Since, in our approach a business rule may be mapped to a set of operations that belongs to an expert object (see ExpertObject in LinkModel) responsible to hold in those operations. Therefore, every object model must be referred by at least one business rule. The grouping of business rules around their responsible object model allows organizing rules and their resulting operations into object model thus avoiding duplication.

7.2.2 Correlation based on a common or opposed Predicate

Business rules can refer to the same predicate or business condition. So, the resulting methods related to their corresponding rules actions must be embedded in the same interaction fragment ALT with the same guard in SDSIB. For example the rules 1, 2 and 7 in Figure 9-a represent a set of rules which are controlled by the common business predicate: "the product's stocked quantity is greater than the item's quantity". This common predicate will represent a guard in the SDSIB interaction fragment where the resulting methods calculatePR-QTY(), calculateIT-SUBTOTAL() and AddItem relative respectively to the rules 1, 2 and 7 will be embedded. In other, if the rules are controlled by business predicates that use the same arguments except that those predicates are opposed such as the predicate of rule 9 which is the negation of the common predicate of rules 1, 2 and 7. In this case the matched method relative to the rule 9 will be embedded in the same interaction fragment ALT but in a different branch controlled by another guard that corresponds to its business predicate.

7.2.3 Correlation based on referencing business result

Some rule results are or used or referenced as an argument in other ones. We call this type of relationship a correlation based on referencing result. A business rule may have more than referencing result correlations if its result is referenced in many other rules. This kind of correlation defines the order of the methods execution in SDSIB. Thus, the business rule that references or uses the result of another business rule involves the apparition of its methods in SDSIB after the other one that references it. The figure 9-b shows for example the rule 1: "subtotal of the ordered article" is

![Fig. 9-a. Business rules correlation and corresponding SDSIB](image-url)
referred in the rule 3 as an argument to calculate the total of command. Thus, the method \textit{calcuiateIT-SUBTO}

\textit{TAL() \textit{relative to the business action of the rule 1 must appears before the method \textit{CalculateCD-TOTAL()} concerning the rule 1. Idem, the matched method \textit{CalculatingCU-Totsales()} related to the rule 5 which calculates the total sales of a given customer must appears in the third rank because it references the total of command. In addition to orderly representing methods in SDSIB, each computational rule that references derived or simple attributes in other domain classes as arguments should call for the getters methods to retrieve their corresponding values already calculated by the other methods.

![Fig. 9-b. Business rules correlation and corresponding SDSIB](image)

### 7.2.4 Correlation based on a common Loop

Business rules can refer to the same loop existing in the action expressions related to a given rule. So, the resulting methods related to their corresponding rules must be embedded in the same interaction fragment Loop in SDSIB within a condition end extracted also from the action expressions. For example the rules 1, 2 in Figure 9-c use the same iterative statement: \textit{“PERFORM UNTIL REP= ‘Y’”}. This condition end will represent a guard in the SDSIB interaction fragment Loop where the resulting methods \textit{calculatePR-QTY()} and \textit{calculateIT-SUBTOTAL()} will be embedded. In other side an interaction fragment can be embedded in another one such as the following example below when an ALT fragment is nested in a loop fragment because the business conditions of both rules are controlled by the same loop statement.

![Fig. 9-c. Business rules correlation and corresponding SDSIB](image)
8. Evaluation

In this section we are in a position to discuss two major points: one concerning a preliminary evaluation of the accuracy of our BRE that use a slicing algorithm to extract business rules and the second point consists of evaluating our methodology and requirements for support tool with previous ones presents in market or developed in laboratories.

Regarding the first point of evaluation, to test the accuracy of our BRE approach supported by the backward static slicing based on PDG, a criterion was defined to check how close are the business rules extracted by our BRE (named BRE results) to the business rules extracted manually (named expected results). Indeed, we have chosen a set of arbitrary programs written in COBOL to test and we have used three evaluation metrics: recall, precision and F-measure.

Table 3 shows the recall, precision and F-measure calculated for many programs. The main challenge in our BRE approach is to simultaneously obtain high precision recall and F-measure. As can be seen from table 3, the calculated average recall, precision and F-measure are equals respectively to 78.41%, 89.28% and 83.39% that are encouraging for initial experiments.

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected BR</th>
<th>Correct extracted BR</th>
<th>Incorrect extracted BR</th>
<th>Missing BR</th>
<th>Recall (%)</th>
<th>Precision (%)</th>
<th>F-measure (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRG 1</td>
<td>10</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>80.00</td>
<td>88.89</td>
<td>84.21</td>
</tr>
<tr>
<td>PRG 2</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>87.50</td>
<td>87.50</td>
<td>87.50</td>
</tr>
<tr>
<td>PRG 3</td>
<td>12</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>75.00</td>
<td>81.82</td>
<td>78.26</td>
</tr>
<tr>
<td>PRG 4</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>90.91</td>
<td>100.00</td>
<td>95.24</td>
</tr>
<tr>
<td>PRG 5</td>
<td>14</td>
<td>9</td>
<td>2</td>
<td>3</td>
<td>64.29</td>
<td>81.82</td>
<td>72.00</td>
</tr>
<tr>
<td>PRG 6</td>
<td>12</td>
<td>10</td>
<td>0</td>
<td>2</td>
<td>83.33</td>
<td>100.00</td>
<td>90.91</td>
</tr>
<tr>
<td>PRG 7</td>
<td>10</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>70.00</td>
<td>87.50</td>
<td>77.78</td>
</tr>
<tr>
<td>PRG 8</td>
<td>11</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>63.64</td>
<td>77.78</td>
<td>70.00</td>
</tr>
<tr>
<td>PRG 9</td>
<td>12</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>91.67</td>
<td>100.00</td>
<td>95.65</td>
</tr>
<tr>
<td>PRG 10</td>
<td>9</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>77.78</td>
<td>87.50</td>
<td>82.35</td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>78.41</td>
<td>89.28</td>
<td>83.39</td>
</tr>
</tbody>
</table>

Regarding the second part in our evaluation, we have examined several BR tools that are available in the market or research area. Firstly, the BR tools can be classified into two main categories, each of which provides some of the features, required for the Business Rules Management Solution (BRMS). The two categories are following:

- Category 1: BR tools for forward engineering.
- Category 2: BR tools for reverse engineering.

The tools which belong to the first category offer various features for forward engineering and focus on business rule domain integration, writing high-level rules, organization in flow-rules or processes, and specifically business rule implementation. Many tools on the market and research belong to this category. For example, WebSphere ILOG JRules BRMS (IBM, 2012) which provides both collaborative rule management for business teams and robust, scalable and precise rule execution. RuleML (H. Boley, 2003) is a XML-based Rule Markup Language that has been proposed by the Rule Markup Initiative as a canonical language for publishing and sharing rules on the Web. RuleML is implemented with XML Schema, XSL Transformations (XSLT). QuickRules (M.D’Hondt, 2004) offers a BRMS for the Java/J2EE platform. It provides an Eclipse-based graphical user interface targeted at domain experts and developers which allows for the design, implementation, testing, and deploying of business rules.

The tools of the second category support knowledge-based legacy application. They facilitate the acquisition of knowledge by extracting business rules, modeling, translating and code implementation of collected BR. Our methodology and requirements that support our tool is comprised in this category of tools. These tools are typically versatile tools offering a wide variety of features for BRMS. Many BR tools fall into this category. For example the SOFT-REDOC developped by (HARRY M, ET AL.,1996) is a reverse engineering tool for COBOL program which extracts business rules embedded in code by manually input of the criteria elements. The Business Rule Extraction Environment tool (BRE) (Huang,
1996-b) parses an input COBOL program and builds a syntax tree and data dictionary, then, dependence analyzer performs data flow analysis on the syntax tree and constructs dependence graphs by means of the PDG and data dictionary. Softwaremining's BRE Toolkit (SoftW, 2010) uses sophisticated search algorithms to sieve through complex program code and isolate the BR embedded within it. The system allows assignment of descriptions in natural language to the set of variables referenced within the identified rule. This feature can be used to reproduce BR in NL form (i.e. pseudo-English). Microfocus Modernization Workbench platform (MicroFocus, 2011) is a comprehensive solution for understanding your application portfolios. It stores the application information in a central location that can be accessed and adapted through a set of modules. Each module provides distinct capabilities that support development and modernization. MagicDraw (Nomagic, 2010) is a modeling tool which supports business process, architecture, software and system aspects with teamwork support. It is designed for business analysts, software analysts and programmers. MagicDraw supports also the UML 2 metamodel, the latest XMI standard for data storage and the most popular programming languages for implementation. Finally, in the following table we will present the different criteria of various BR tools that falls into the second category.

**Tab. 3: Comparison study of BRMS tools of the second category**

<table>
<thead>
<tr>
<th>tools</th>
<th>Extracting BR</th>
<th>detection relevant variables</th>
<th>support meta-models</th>
<th>Translation to NL</th>
<th>incorporate SDSIB interactions</th>
<th>Supporting implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR Extraction Environment</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>SOFT-REDO</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Softwaremining BRE Toolkit</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Microfocus Modernization Workbench</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>MagicDraw</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Our proposal</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

9. **Conclusion and perspectives**

Business rules, which can be used to represent both systems requirements and conditions to which the system should constrained. They are considered as the most relevant part in software applications. However, due to their incessant changes according to the new legislation or enterprise development, they present a huge impact on both the business processes and the software itself. In this paper we have presented an approach for business rule management which aims to capture, model, store and represent business rules in an accessible form that enables both decision-makers and programmers to efficiently benefit from them. The proposal establishes a bridge between business logic and application code by means of translating the business rules in natural language and generating objects operations and interactions in an object-oriented platform.

Our approach joins the topic specialized in applying MDE to automate reverse engineering process and it presents many advantages to practitioners and researchers from industry and academia with a vested interest in this area of reuse and modernization of legacy systems. This work completes also our previous ones (El Beggar, et al., 2013b), which interested in extracting legacy data and representing it in advanced platforms like relational database. Therefore, we have attempted to present a methodology for migrating legacy systems to new scalable ones which are independents of platforms and protected from any technology rupture. Furthermore, the proposal reduces the systems time re-engineering given that MDRE is a generative approach based on a set of model transformations. We have proposed also a comparative study between classical methodologies of extracting knowledge from legacy systems and our proposal MDRE (El Beggar, et al., 2013a). Finally,
the paper (El Beggar, 2012) which aims to generate code for Java platform from SDSIB model is an effective extension of our proposal.

However, we are actually engaged in further refinements to improve our approach presented in this paper. Firstly, a number of case studies of different domains are currently being used to test our approach. The study of the others types of business rules taxonomy will also play a significant contribution to metamodel refinement and improve our approach to cover all kind of business rules. Secondly, to improve our MDRE approach another enviable perspective which consists of translating the natural language expressions to formal languages such as Java and C# can be also a prospect to initiate. Additional components that contribute to this purpose must be added to the mapping metamodels or further metamodels will be proposed. Finally, the CASE tool that supports our approach is currently under development.

REFERENCES


Choi, J. D. and Ferrante, J., 1994: Static slicing in the presence of goto statements. ACM Transactions on Programming Languages and Systems, 16(4):1097–1113


Chikofsky, E. J. and Cross, J. H., 1990: Reverse engineering and design recovery: a taxonomy, IEEE Software 7 (1) 13–17
EL BEGGAR O., BOUSETTA, B. and GADI, T., 2012. Automatic code generation by model transformation from sequence diagram of system’s internal behavior”. International Journal of Computer and Information Technology (IJCIT) ISSN: 2279 – 0764, 1(2), p130-146

EL BEGGAR O., BOUSETTA B. GADI, T. 2013a, Comparative Study between Clustering and Model Driven Reverse Engineering Approaches. The 5th International Conference on Computer Engineering and Technology ICCET 2013, Vancouver, Canada.


**JEL Classification: M15**