

*Semiautomatic Reverse Engineering Tool on Oracle  
Forms Information Systems*

OSCAR JAVIER CHAPARRO ARENAS

CODE: 02300403



UNIVERSIDAD NACIONAL DE COLOMBIA  
FACULTAD DE INGENIERÍA  
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS E INDUSTRIAL  
BOGOTÁ, D.C.  
DECEMBER 2012

*Semiautomatic Reverse Engineering Tool on Oracle  
Forms Information Systems*

OSCAR JAVIER CHAPARRO ARENAS

CODE: 02300403

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE OF  
MASTER IN ENGINEERING - SYSTEMS AND COMPUTING

ADVISOR

JAIRO APONTE, PH.D

RESEARCH LINE

SOFTWARE ENGINEERING

RESEARCH GROUP

COLSWE: SOFTWARE ENGINEERING RESEARCH GROUP



UNIVERSIDAD NACIONAL DE COLOMBIA  
FACULTAD DE INGENIERÍA  
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS E INDUSTRIAL  
BOGOTÁ, D.C.  
DECEMBER 2012

**Title in English**

Semiautomatic Reverse Engineering Tool on Oracle Forms Information Systems

**Título en español**

Herramienta de ingeniería inversa semiautomática sobre sistemas de información legados en Oracle Forms.

**Abstract:** Legacy information systems are systems that have had a long evolution, longer than the typical turnaround time of the developers in the company. They are essential to the business and encode large amounts of essential information related to the business processes. However, continuous changes in the domain of some systems, the lack of strict maintenance processes, and the turnaround of developers inevitably leads to gradual loss of knowledge about the system and its domain, most often corroborated by the fact that external documentation is rarely updated in synch with the code and other artifacts. SIFI, a financial legacy information system owned by the software development company IT Consultores S.A.S., presents such a maintenance issues: high coupling, architecture decay, no formal documentation and loss of domain and implementation knowledge, making its evolution very difficult. Within this thesis, a reverse engineering tool for Oracle Forms and PL/SQL information systems was built, aiming at supporting the maintenance process on SIFI. The tool is able to extract and visualize structural and behavioral information about the system, and implements the approach we proposed for automatically extracting structural business rules from legacy databases. The effectiveness of the tool was assessed under the understanding and maintenance of SIFI, through a survey. The results show that tool is very useful, as it improves the productivity of developers to complete their tasks and the maintenance process of SIFI is now easier for them. In addition, the implemented business rule extraction approach was assessed though a study with 4 ITC employees. The results show that the recovery technique is practical, while there is room for improvement, and it will be used as basis for the recovery of additional knowledge.

**Resumen:** Los sistemas legados son sistemas que han tenido una larga evolución, más larga que el tiempo típico de los desarrolladores en una empresa. Estos sistemas son esenciales para el negocio y contienen grandes cantidades de información sobre los procesos de negocio. Sin embargo, los cambios continuos en el dominio de algunos sistemas, la falta de procesos estrictos de mantenimiento, y los cambios de desarrolladores conducen de manera inevitable a pérdidas graduales de conocimiento del sistema y su dominio, lo cual es corroborado por el hecho de que la documentación externa es rara vez actualizada, de acuerdo con el código y otros artefactos. SIFI, un sistema de información legado desarrollado y mantenido por la empresa de desarrollo de software IT Consultores S.A.S, presenta tales problemas de mantenimiento: alto acoplamiento, decaimiento de la arquitectura, sin documentación formal y con pérdida de conocimiento acerca de su dominio e implementación, lo cual hace que su evolución sea difícil. En esta tesis se construyó una herramienta de ingeniería inversa para sistemas de información en Oracle Forms y PL/SQL, con el objetivo de apoyar el proceso de mantenimiento de SIFI. La herramienta es capaz de extraer y visualizar información estructural y comportamental del sistema, e implementa la técnica que hemos propuesto para extraer automáticamente reglas de negocio estructurales de bases de datos legado. A través de una encuesta se evaluó la efectividad de la herramienta considerando el mantenimiento y entendimiento

de SIFI. Los resultados muestran que la herramienta es muy útil porque mejora la productividad de los desarrolladores en completar sus tareas y ahora el proceso de mantenimiento de SIFI es menos complicado. Asimismo, la técnica de extracción de reglas de negocio fue evaluada a través de un estudio con 4 colaboradores de ITC. Los resultados muestran que la técnica es práctica, habiendo posibilidad de mejora, y será usada como base para recuperar información adicional.

**Keywords:** Reverse Engineering, Legacy Information Systems, Business Rule Extraction, Software Maintenance, Software Understanding

**Palabras clave:** Ingeniería Inversa, Sistemas de Información Legados, Extracción de Reglas de Negocio, Mantenimiento de Software, Entendimiento de software

## Acceptation Note

Thesis Work

Approved

“Laureate mention”

---

Jury

Andrian Marcus, Ph.D.

---

Jury

Massimiliano Di Penta, Ph.D.

---

Advisor

Jairo Aponte, Ph.D.

Bogotá, 10/12/2012

---

---

## Dedication

---

---

This Thesis is dedicated to my parents, professors, fellows and other people who helped me in different ways to complete this project.

---

---

## Acknowledgments

---

---

For me, this project was an enriching experience from every perspective. From the professional point of view I acquired new skills in the development of a project and a product, from a research standpoint, I was able to address a problem in a systematic and rigorous way; and from the personal standpoint, this project included a high learning factor regarding the relationship with fellows, colleagues and other people involved.

First of all, I want to thank my parents for helping me to undertake this master project and to take me to new and deep horizons in my life. They will be in my heart forever. I also wish to thank my fellows and professors from the university, including my advisor and friends from the research group, for their ideas, advices, time and unconditional support in the development of this thesis. They are the best people I've ever met.

Additionally, my sincere thanks to all the people of ITC, including managers, developers and functional people. This thesis was to support the company but I think their help was much more than mine. I would like to give special thanks to those in ITC who believe applied research as a means of solving practical problems, and for those who have a vision of creating a better world, from a small dimension such as computing.

I want to thank the National University of Colombia and all its people, for contributing to a better country, through the support to research projects. I am happy to belong to this university and much more to contribute to the development of computing, specifically to software engineering, even if it is a minimal contribution. Finally, there too much work to be done in this field. Every day I will be more thirsty for solving research and engineering problems.

Thank you all.

Oscar Chaparro

Bogotá, November 2012

---

---

# Contents

---

---

<b>Contents</b>	<b>I</b>
<b>List of Tables</b>	<b>IV</b>
<b>List of Figures</b>	<b>V</b>
<b>1. Introduction</b>	<b>1</b>
1.1 Background and justification . . . . .	1
1.2 Problem definition . . . . .	3
1.3 Thesis Organization . . . . .	4
<b>2. Reverse engineering and Business Rules Extraction</b>	<b>5</b>
2.1 Reverse Engineering concepts and relationships . . . . .	6
2.1.1 Reverse Engineering and Software Comprehension . . . . .	6
2.1.2 Reverse Engineering and Software Maintenance . . . . .	6
2.1.3 Reverse Engineering concepts . . . . .	7
2.2 Techniques in Reverse Engineering . . . . .	8
2.2.1 Standard techniques . . . . .	8
2.2.2 Specialized techniques . . . . .	9
2.2.2.1 Programming plans matching . . . . .	10
2.2.2.2 Execution traces analysis . . . . .	10
2.2.2.3 Module extraction . . . . .	12
2.2.2.4 Text processing . . . . .	12
2.3 Business Rules Extraction . . . . .	13
2.3.1 Manual BRE . . . . .	13
2.3.2 Heuristic BRE . . . . .	14
2.3.3 Dynamic BRE . . . . .	15



---

2.4	Summary . . . . .	15
<b>3.</b>	<b>Automatic Extraction of Structural Business Rules from Legacy Databases</b>	<b>16</b>
3.1	Business Rules . . . . .	17
3.1.1	Definition of Business Rules . . . . .	17
3.1.2	Structure of Business Rules . . . . .	17
3.1.3	Types of Business Rules . . . . .	18
3.1.4	What are not Business Rules? . . . . .	19
3.1.5	The Relation between Business Rules and Software Information Systems . . . . .	20
3.2	Extraction of Structural Business Rules . . . . .	21
3.2.1	DB - BR Mappings . . . . .	21
3.2.2	BR Format and Schema . . . . .	23
3.2.3	BRE Algorithm . . . . .	25
3.2.4	PP Module Specific Heuristics . . . . .	26
3.3	Summary . . . . .	28
<b>4.</b>	<b>ReTool: An Oracle Forms Reverse Engineering Tool</b>	<b>29</b>
4.1	Oracle Forms Technology . . . . .	30
4.2	ReTool's Architecture . . . . .	31
4.3	ReTool's Components and features . . . . .	32
4.3.1	Extractors and Analyzers . . . . .	32
4.3.2	Repository . . . . .	34
4.3.3	Visualizers . . . . .	38
4.3.4	Utilities . . . . .	50
4.4	Further details about ReTool . . . . .	52
4.5	Limitations and future enhancements . . . . .	53
4.6	Summary . . . . .	54
<b>5.</b>	<b>Evaluation and Discussion</b>	<b>55</b>
5.1	Evaluation of ReTool . . . . .	55
5.1.1	Purpose of the survey . . . . .	55
5.1.2	Subjects . . . . .	55
5.1.3	Survey description . . . . .	56
5.1.4	Results . . . . .	56

---

5.1.5	Analysis and Discussion . . . . .	61
5.2	Evaluation of the BRE technique . . . . .	63
5.2.1	Evaluation . . . . .	64
5.3	Summary . . . . .	67
<b>6.</b>	<b>Conclusions</b>	<b>68</b>
6.1	How does ReTool support the maintenance and understanding of SIFI? . . .	68
6.2	Future Work . . . . .	69
6.3	Recommendations about the maintenance of SIFI . . . . .	70
	<b>ReTool Evaluation survey</b>	<b>71</b>
	<b>Bibliography</b>	<b>74</b>

---

---

## List of Tables

---

---

3.1	Summary of the heuristics used in the BRE algorithm. . . . .	27
5.1	Examples of the extracted BR components from the PP module. . . . .	64
5.2	Statistics of the extracted BRs and components from the PP module. . . . .	65
5.3	Results of the BR evaluation. . . . .	65
5.4	Quantitative information about the misunderstood/incorrect rules. . . . .	66

---

---

## List of Figures

---

---

2.1	Reverse Engineering process according to [36]. . . . .	7
2.2	Modularization process based on clustering/searching algorithms [38]. . . . .	12
2.3	Business rules knowledge through abstraction levels. . . . .	13
3.1	Business rules structure. . . . .	18
3.2	Element of guidance metamodel [44]. . . . .	20
3.3	Relationship between information systems and business rules. . . . .	20
3.4	DB - BR component mappings. . . . .	24
3.5	Implemented BR database model. . . . .	25
4.1	Example of a form module of an Oracle Forms application. . . . .	30
4.2	General architecture of an Oracle Forms application [3]. . . . .	31
4.3	General architecture of ReTool. . . . .	32
4.4	Command-line options of the program DB Analyzer. . . . .	33
4.5	Comparison of the structure of the form CUSTOMERS and output of the Forms Parser. . . . .	34
4.6	Comparison of the elements of the item COMMENTS and the generated XML file by the Forms Parser. . . . .	35
4.7	Command-line options of the program Forms Analyzer. . . . .	36
4.8	Command-line options of the program Dependencies Analyzer. . . . .	36
4.9	Example of the repository metamodel, which represents the database tables and their relational components, of the target system . . . . .	37
4.10	Graphical User Interface of ReTool Web. . . . .	38
4.11	Graphical elements of a box. . . . .	39
4.12	Pop-up dialog displayed for a table column FOND_DESCRI. . . . .	40
4.13	Icons of every type of object in ReTool Web. . . . .	41
4.14	ReTool Web's login page. . . . .	41

---

4.15	Searching page. . . . .	42
4.16	Searching results page. . . . .	43
4.17	Object References page. . . . .	44
4.18	Table References page. . . . .	45
4.19	Form References page. . . . .	45
4.20	Table Hierarchy page. . . . .	46
4.21	Dependencies Tree page. . . . .	47
4.22	Call Sequences page. . . . .	47
4.23	Package Information page. . . . .	48
4.24	Tables-Form page. . . . .	48
4.25	Forms-Table page. . . . .	49
4.26	SQL Inserts Generation page. . . . .	50
4.27	Reverse Engineering Processes page. . . . .	51
5.1	Distribution of the survey respondents by working group. . . . .	56
5.2	Distribution of the survey respondents by working time. . . . .	57
5.3	Level of feature usage of the tool. . . . .	57
5.4	Level of tool usage on development tasks. . . . .	58
5.5	Degree of tool support in reducing the time to complete tasks. . . . .	59
5.6	Degree of tool support in reducing the effort to complete tasks. . . . .	59
5.7	Degree of tool support in increasing the precision to complete tasks and artifacts. . . . .	59
5.8	Rating of some quality attributes of the tool. . . . .	60
5.9	Status of the received suggestions from the respondents. . . . .	60
5.10	Specific categorization of the non provided suggestions. . . . .	61
5.11	Categorization of the non provided suggestions. . . . .	61

---

---

## Introduction

---

---

### 1.1 Background and justification

Software evolution is the changing process of software through time. This process includes the conception phase, the development and maintenance stages, and the phase-out of the software project and product. Software evolution is an inevitable process because of the environmental changes, the new emerging concepts and business rules, and the evolution of technology regarding hardware, software, programming and software engineering paradigms [9]. From this perspective and depending on the evolutionary strategies applied to software [9], every software system will eventually be legacy in the future.

Legacy software systems are those that have had a long evolution. These systems were developed with outdated technologies and paradigms [27][42] and have a large size in terms of information processing capacity, functionality provided, program components, lines of code (LOC) and other size metrics. They are often difficult to modify and have low integration level with other systems [42]. On the other hand, they are essential for business [9][27] and encode large amounts of essential information related to the business processes.

The characteristics mentioned above have implications on the evolution of those software systems. Continuous changes in the system's domain, the lack of strict maintenance processes, and the turnaround of developers inevitably leads to gradual loss of knowledge about the system (its structure and behavior) and its domain, most often corroborated by the fact that external documentation is rarely updated in synch with the code and other artifacts. Consequently, one of the most important problems in the evolution of legacy systems is the lack of knowledge about them [27]. The most up to date source of information for recovering this knowledge is the structure, the behavior and the data of the information system.

These problems have been addressed by several software reverse engineering techniques and methods [13][27]; Canfora Harman *et al.* [13] evidence the work and the success in the area. Reverse engineering is defined by [14] as the analysis of a system in order to identify its components, relationships and behavior. The field aims at creating system representations in other forms or at higher abstraction level independent of the implementation. The main goal of the field [13][47] is supporting software comprehension/understanding,

software maintenance and software re-engineering. More specifically [14][47], reverse engineering is employed to reduce software complexity, to generate alternative software views from different architectural perspectives, to retrieve “hidden” information as a consequence of system long evolution, to detect side effects or non-planned design branches, to synthesize software through the generation of high level abstractions, to facilitate software reuse, to assess correctness and reliability of software systems, and to locate, trace and evaluate defects or changes quickly.

Additionally, software reverse engineering is a process that consists of four steps [47]: data extraction from software artifacts, data processing and analysis, knowledge persistence in a repository and presentation of knowledge. The first step is the acquisition of raw data from several sources of information of the software system. Subsequently, in the second step, these data are subjected to cleaning, filtering, transforming and structural, semantic and behavioral analyses. The resulting information of this analysis is stored in a repository or a knowledge database; and finally, this information is presented to the user in a graphical or textual way through different software views. In the first step, the main source of information is the source code, however, there are other sources that are complementary to this type of information. Specifically, there are three types [13]: static sources, such as the source code or the system documentation (in general, any other information that is not produced by the system execution), dynamic sources, which provide information resulting from the system execution such as execution traces; and historical sources or information of the system evolution, such as version control repositories.

Reverse engineering is a process of knowledge extraction, where such knowledge is useful and necessary for other processes. Indeed, its importance lies in the role it has in other domains where this method is necessary. Examples include software maintenance and migration. Software maintenance is costly because of the essential properties of the software; reverse engineering is required for maintaining legacy systems since they are large and complex, and difficult to modify. In software migration, reverse engineering is the first step and is used to extract representations of a system at different levels of abstraction and independent of the implementation. The second step is re-engineering, in which design transformations are made, leading to new representations and a new implementation of the system. Sometimes there are merely programming language transformations.

This thesis describes the application of reverse engineering on SIFI (SIstema Fiduciario Integrado<sup>1</sup>), which is an information system implemented in PL/SQL and Oracle Forms technology, and has a life cycle of over fifteen years. The information system is developed in Oracle Forms, an outdated (obsolete) technology that has several limitations from the evolutionary point of view. Oracle Forms is an enterprise application technology that provides a two-layer distributed client-server architecture [55]. The business logic in this technology is implemented on both client and server through the PL/SQL language, a language extension to SQL that provides procedural programming to the Oracle database technology. This type of two-layer architectures has some advantages over one-layer ones [55], for example, there is a load distribution among clients, the job of the server is executed under only one context and the performance of the system is often high because of the high coupling possibility of the system. However, these advantages have some limitations from the evolutionary point of view [2][55]:

---

<sup>1</sup>In English, Fiduciary/Trust Integrated System

1. Scalability issues due to the centralized nature of the server and decentralization of the clients. A modification in the system often implies a large update in all the clients.
2. Strong coupling since the business logic can be integrated (mixed) with the view (the graphical user interface) and data management layers, and therefore, the business logic is difficult to isolate and change.
3. Strong dependence on the tools' vendor, which is risky if the vendor stops providing support. Also, these low-layer architectures have very poor portability.

Furthermore, SIFI has a large complexity and size, and therefore, its maintenance, understanding and evolution are relatively difficult. SIFI is a financial information system that manages the data and the operations related to investments funds, financial investments, trusts, accounting, budget, treasury, accounts payable, billing and commissions portfolio. The system is maintained by a Colombian software development company (i.e., IT Consultores S.A.S. or just ITC) and has been deployed for more than fifteen years, operating in several trust companies (banking companies). It is currently deployed and used in nine large Colombian trust companies that represent 60% of this market in the country. The system is developed using Oracle Forms, has 1.400 form modules, 3.200 database tables, 3.500 stored procedures, 700 database views and around 800 KLOC in the database.

## 1.2 Problem definition

SIFI has some maintenance issues that make it difficult to change and evolve. Some of these are:

- SIFI is implemented in Oracle Forms 6i, an obsolete technology that has several maintenance restrictions (described briefly above).
- The system has high coupling between structural objects, such as database tables, and behavioral objects, such as database packages and procedures.
- The architecture of the system has decayed over time, mainly because of a uncontrolled maintenance process.
- The system has no formal documentation about technical and business/domain aspects of the system. There are no traceability links between artifacts either.
- The knowledge about the system has remained mainly within two groups of people: the technical group, who knows SIFI's architecture, and the business group, who knows the business processes supported by SIFI. The problem is that when people leave the company, a loss of knowledge about the system and the business occurs.

Then, the problem addressed by this thesis is synthesized in two questions:

- How can we support SIFI's maintenance and understanding processes?
- How can we extract business information from SIFI, such as business rules?



In this sense, a reverse engineering process is required to tackle these problems, in such a way that structural, behavioral and business information of the system is extracted from the source code and other sources of information. The resulting information and knowledge of this process will be oriented to support the evolving tasks on the system.

The main contribution of this thesis is to create a reverse engineering tool for Oracle Forms and PL/SQL information systems. Specifically, reverse engineering techniques are implemented for the extraction of system structural and behavioral information, and a Business Rules Extraction technique is proposed and developed. The effectiveness of the tool to support the understanding and maintenance of SIFI was confirmed by the evaluation carried out. Moreover, this work resulted in a list of recommendations about how the maintenance process on SIFI should be addressed by the company.

The reason for creating a tool is that the company is interested in obtaining all the possible knowledge from this reverse engineering process, considering the implications of performing this process and the fact that the company is planning to migrate SIFI to a web technology.

### 1.3 Thesis Organization

This document is structured as follows:

- Chapter 2 presents the concepts, methods, and motivation for performing reverse engineering and business rules extraction.
- Chapter 3 describes the proposed method for extracting business rules from legacy databases.
- The reverse engineering tool we created is described in detail in Chapter 4.
- Chapter 5 discusses the results of the tool evaluation. In the same way, the evaluation results of the business rule extraction technique are described thoroughly.
- Finally, this document presents the conclusions, future work and recommendations about the maintenance process on SIFI in Chapter 6.

---

---

# Reverse engineering and Business Rules Extraction

---

---

Software Reverse Engineering is a field that consists of techniques, tools and processes to get knowledge from software systems already built. The main goal of this discipline is to retrieve information about how a software system is composed and how it behaves according to the relations of its components [14]. Reverse Engineering aims at creating representations of software in other forms or in more abstract representations, independent from the implementation, e.g., visual metaphors or UML diagrams. Techniques in the field are employed for many purposes [14]: to reduce system complexity, to generate alternative views of software from several architectural perspectives, to retrieve “hidden” information as a consequence of a long evolution of systems, to detect side effects or non-planned design branches [47], to synthesize software, to facilitate reuse of code and components, to assess the correctness and reliability of systems, and to locate and track defects or changes faster.

Reverse Engineering (RE) arises as an important area in software engineering, since it becomes necessary in Software Evolution. Software Evolution is an inevitable process, basically because most factors related to software (and technology) change [9]: business factors such as business concepts, paradigms, processes, etc., and technology factors such as hardware, software, software engineering paradigms, etc. RE is especially important in regard to legacy systems since they suffer degradation and have a long operational life, producing a loss of knowledge about how they are built. Generally, the changes in this kind of software actually happen but are not well-documented. Instead, knowledge about changes is kept by people, but people are volatile: people leave projects and companies, and people forget easily. Therefore, knowledge needs to be extracted directly from software (source code) and its behavior (run-time information), which are both the main sources of information for RE. In this sense, the general problem is how to extract information or knowledge from software artifacts.

Reverse Engineering can be considered as a more general field compared to Software Comprehension, as the first one involves more field actions and not only comprehension of code (e.g., RE is also used for fault localization), although sometimes the comprehension process is a secondary result of it. The philosophy about RE is the application of techniques to know how software works, how it is designed and how this design allows software to

behave the way it does. Consequently, in this process it is perfectly natural that the comprehension part appears as an indirect consequence or as a motivation.

Since Reverse Engineering is vital for software development processes, this chapter presents an overview of some techniques in the area. The chapter is organized as follows: in section 2.1, the needs, benefits and purposes of RE are reviewed, in the context of Software Understanding and Maintenance. Section 2.2 presents a review of some techniques in the field.

## 2.1 Reverse Engineering concepts and relationships

The term Reverse Engineering has been used to refer to methods and tools related to understanding (or comprehending) and maintaining software systems. RE techniques have been used to perform systems examination, so in Software Understanding and Maintenance, RE has been a useful medium to support these processes.

### 2.1.1 Reverse Engineering and Software Comprehension

Software Comprehension is the process performed by an engineer or a developer to understand how a software system works internally. The understanding process involves the comprehension of the structure, the behavior and the context of operation of a program. Along with these attributes, the explanation of problem domain relationships is required [15]. Understanding is one of the most important problems in Software Evolution; it is said that between fifty and ninety percent of the effort in maintenance stages is devoted to this task [40]. Commonly, system documentation is out of date, incorrect or even inexistent, which increases the difficulty of understanding what the system does, how it works and why it is coded that way [8].

Several comprehension models and cognitive theories are reviewed by Storey in [49]. It could be said that the main models, or at least the most common, are top-down and bottom-up. Top-down comprehension strategy is basically the mapping between previous system/domain knowledge and the code, through formulation, verification and rejection of hypotheses. In terms of the implementation of a RE tool, this process typically includes rule matching to detect how code chunks achieve sub-goals within a specific feature or plan [43]. When performing automatic RE to legacy software, bottom-up approach is commonly used because top-down approach requires detailed knowledge about the “goals the program is supposed to achieve” [11]. In bottom-up understanding, software instructions are taken to infer logic and semantic groups, categories and goals. The automation of this approach is very complex [11][43] and is not supposed to be solved by a single technique, because of the semantic gap between code and domain knowledge of a system. Additionally, developers actually need a variety of functionalities and information that one technique or implementation may not achieve or provide.

### 2.1.2 Reverse Engineering and Software Maintenance

Software Maintenance is usually defined as the process made on software after its delivery to production environment [12]. Common activities in maintenance are correction of defects, performance improvement and software adaptation due to changes in requirements

or business rules. Software Maintenance is divided into 4 categories [50]: Corrective maintenance, Adaptive maintenance, Performance enhancement and Perfective maintenance.

Reverse Engineering comprises the first step in Software Maintenance: the examination activity. Changes in software are executed later; therefore, RE does not involve the changing process [12].

### 2.1.3 Reverse Engineering concepts

Software Reverse Engineering was defined by Chikofsky *et al.* [14] as the process of analyzing a system to:

- Identify the system's components and their inter-relationships, and
- Create representations of the system in another form or at a higher level of abstraction.

A discussion of this **definition** is developed in [51]. In this work, authors state that this definition does not fit to all techniques in the field; for example, *program slicing* does not recover system's components and relationships. This definition does not specify what kinds of representations are considered and the context in which the process is executed, so the role of automation and the knowledge acquisition process are not clear. In this sense, the authors propose a more complete definition: "Reverse Engineering includes every method aimed at recovering knowledge about an existing software system in support to the execution of a software engineering task".

The **process of Reverse Engineering** is divided into four phases [36]: Context Parsing, Component Analyzing, Design Recovering and Design Reconstructing. Figure 2.1 shows the whole process.

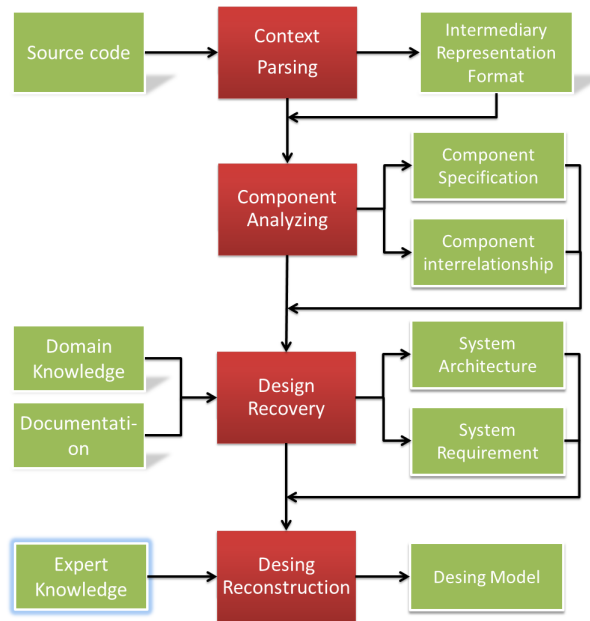


FIGURE 2.1. Reverse Engineering process according to [36].

Asif in [4] presents the elements involved in the Reverse Engineering process:

- Extraction at different levels of abstraction,
- Abstraction for scaling through more abstract representations,
- Presentation for supporting other process such as maintenance, and
- User specification allowing the user to manage the process, the mappings for transformation of representations, and software artifacts.

Khan *et al.* [47] defines the general process of Reverse Engineering, which consists in four phases: data extraction from software artifacts, data processing and analysis, knowledge persistence in a repository, and presentation of knowledge. The authors also present some **benefits and applications** of Reverse Engineering. RE is used:

- To ensure system consistency and completeness with specification.
- To support verification and validation phases.
- To assess the correctness and reliability of the system in the development phase, before it is delivered.
- To trace down software defects.
- To evaluate the impact of a change in the software (for estimating and controlling the maintenance process).
- To facilitate the understanding by allowing the user to navigate through system in a graphical way.
- To speed up the Software Maintenance and Understanding. A requirement of a RE tool is the fast and correct generation of cross-reference information and different representations of software.
- To measure re-usability through pattern identification.

RE embraces a broad range of techniques, from simple ones, such as call graphs extraction, to more elaborated ones, such as architecture recovery. The trends go towards more sophisticated and automatic methods. The next section presents some techniques in the field.

## 2.2 Techniques in Reverse Engineering

Methods or techniques (used indistinctly) in Reverse Engineering are automatic solutions to one or more problems in the field [51]. This section provides a partial revision of techniques, divided into two categories: standard and specialized techniques.

### 2.2.1 Standard techniques

Standard techniques include mostly basic descriptive techniques, such as dependency models or structural diagrams. The objective of standard techniques is to obtain system structure and dependencies at different levels of abstraction, by applying basic source code

analysis and Abstract Syntax Tree (AST) <sup>1</sup> processing. According to [47], a RE tool typically provides the following views:

- **Module charts:** they present relationships between system components. A module is a group of software units based on a criterion. Theoretically, modules must have a well-defined function or purpose.
- **Structure charts:** in a general sense, they present software objects categorized and linked by some kind of relationship, generally, method calls. According to [57]<sup>2</sup>, these diagrams also show data interfaces between modules. Examples of structure charts are entity-relationship models and class diagrams. Actually, module charts are structure charts.
- **Call graphs:** they describe calls and dependencies between objects at different levels of granularity. These diagrams are created by analyzing AST from code. The level of granularity is set (for instance functions, methods, classes or variables) and then, the AST is traversed to find the usage of objects. Call graphs are important for change propagation analysis.
- **Control-flow diagrams:** at low/medium levels of granularity they present the systems execution flow, in which control structures (e.g., IF or FOR / WHILE) guide the flow. Another diagram of this type is the Control Structure Diagram (CSD) [20], in which source code is enriched through several graphical constructs called “CSD program components/units”, to improve its comprehensibility.
- **Data-flow diagrams:** they are graphs that show the flow of data (in parameters and variables) through features, modules, or functional processes [57].
- **Global and local data structures, and parameter lists:** these allow going to a fine-grained level of software.

The automatic extraction of these diagrams involves several operational tasks on code and its AST. For example, in data flow diagrams, the transformation and the storing of data must be obtained; in this case every parameter needs to be tracked between and inside methods or functions to know what operations include them and how they are used. Before this process, modules need to be determined. In addition, some of these diagrams are the source of information for techniques such as dependencies analysis of code and data, and the evaluation of the changes impact in code.

### 2.2.2 Specialized techniques

Other techniques, called specialized techniques in this chapter, include operations on software artifacts that are intended not only to describe software but to extract knowledge from it. Programming plans matching using artificial intelligence techniques [11], execution traces processing [22][28], module extraction [37][38], natural text processing [42], pattern extraction [24], and Business rules extraction [41][42], are some examples. Some of them are described briefly in this section.

<sup>1</sup>An AST is a tree representation of the syntactic structure of source code. For example, AST View is an Eclipse plugin for visualizing AST for java programs: <http://www.eclipse.org/jdt/ui/astview/>.

<sup>2</sup>See chapter 15 of [57]: Additional Modeling Tools.

### 2.2.2.1 Programming plans matching

One problem in Reverse Engineering is how to find the semantics (the meaning of) of code. In the automation of this process the analysis of code identifiers and concept extraction are almost required. However, another approach is making matches between chunks of code and programming plans stored in a repository [11][43]. A programming plan is a "design element in terms of common implementation patterns" [43]. A plan can be considered as a programming pattern or template, and can be generic or domain-specific. Some examples of plans are READ-PROCESS-LOOP and READ-EMPLOYEE-INFO-CALCULATE-SALARY; the former is a generic plan which means "reading input values and perform some actions to each value" at implementation level. The latter is a specific domain plan that is a specialization of the first one because at implementation level is almost the same as the first plan, but has a more semantic stereotype. This means that the repository has a set of generic and specialized plans organized in a hierarchy. In [43], the authors state that the recognition of programming plans against information of the AST of code is better, in terms of searching cost, if the plan library is highly organized, each plan has indexing and specialization/implication links to other plans<sup>3</sup>.

However, the matching task is a NP problem, so the process is computationally expensive [11]. The work presented in [11] addresses this problem by applying artificial intelligence techniques. The approach is a two-step process. The first step is a Genetic Algorithm execution to make an initial filtering of the plan library based on "relaxed" matching between code chunks [10] and programming plans stored in the library (the repository). The second step uses Fuzzy Logic to perform a deeper matching. The output of the whole approach is a ranked set of programming plans according to a similarity value with a chunk of code.

In summary, the objective to be achieved by these works is to find programming plans similar to a portion of code. Programming plans are stereotypical patterns of code (generic or domain-specific patterns), therefore it is possible to assign high level concepts to programs, once the matching process has been performed.

### 2.2.2.2 Execution traces analysis

As a complement to Static Analysis in RE, processing of run-time information is commonly used in what is called Dynamic Analysis. Maybe the most common source is the system execution trace.

Execution traces analysis refers to execution trace processing to find patterns in traces that have a specific function. The advantage of traces is that they show the portions of software that are being executed in a specific execution scenario. In this way, the search space is smaller than the one in static analysis because the executed portions of code are the only ones considered.

Two problems are detected in dynamic processing: first, knowledge of the system is required to perform this analysis, and second, this dynamic analysis produces huge amounts of information (long traces). The former problem refers to the fact that it is not possible to capture the (infinite) entire execution domain of a system. If there is no knowledge about how the system works, using and executing all its functionalities is not

<sup>3</sup>According to Quilici [11], a plan consists of inputs, components, constraints, indexes and implications.

possible. If it is not necessary to know about the entire system, and the knowledge about the use of specific functionality actually exists, this would not be a problem. The latter problem depends on how the software is built at low level, how the code is instrumented<sup>4</sup> and how much information the user needs. Other problems related to dynamic analysis are low performance, high storage requirement and cognitive load in humans [15].

Object-oriented software has been the most common object of study in traces analysis. For example, in [22], the problem of identifying clusters of classes is addressed based on a technique that reduces the “noise” in execution traces through the detection of what the authors call “temporally omnipresent elements”, which represent execution units (groups of statements) distributed in the trace. In this sense, noise represents information that is not specific to a behavior of interest. For this, samples of a trace are taken and the distribution of each element along the samples is calculated through a measure of temporal occurrence. To cluster elements, dynamic correlation is used. Two elements are dynamically correlated if they appear in the same samples, so the measure of correlation is based on the number of samples in which they occur. The clustering part takes all elements whose correlation is higher than a fixed threshold, thus grouping elements in components.

In summary, the noise of traces is removed and then clustering is applied to the filtered traces. That work presents an industrial experiment of the approach. The system of study was a two-tier client-server application: the client was a Visual Basic 6 application of 240 KLOC and the server was comprised of 90 KLOC of Oracle PL/SQL (Procedural Language/Structured Query Language) code. The client code was instrumented since no trace generation environment was found, and in the case of the server, Oracle tracing functions were used. The system was executed over a use-case, producing a trace of 26.000 calls.

Similarly, the authors in [28] define “utility element” as any element in a program that is accessed from multiple places within a scope of the program. The objective of the authors is to remove these utility elements or classes from the analysis. This is achieved by calculating the proportion of classes that call each other (fan-in analysis) iteratively by reducing the analysis scope or by applying the technique on a set of packages. The utility-hood metric,  $U$ , of the class  $C$  is defined as

$$U = |IN| / (|S| - 1) \quad (2.1)$$

where  $S$  is a set of classes considered in the analysis and  $IN$  is a subset of classes that use  $C$ . Besides this metric, the standard score (z-score) was considered to determine possible utility classes: classes with large and positive z-score values are possible utilities. Once the filtering is performed, the depiction of components is done by a tool that generates Use Case Maps<sup>5</sup>. In this latter step, calls between classes and conditions of execution (control-flow statements) are considered.

As it is noticed, the main challenge in execution trace analysis is how to reduce and process the trace, so the final result is a good abstraction of what a system does under a specific execution scenario. The main problems to be addressed are the definition of the information that the traces should have, the metrics and procedures that should be used

<sup>4</sup>Code instrumentation refers to the use of software tools or additional portions of code in the system, through which execution traces and behavior information of the system are gathered.

<sup>5</sup>For more information about this type of models refer to [www.usecasemaps.org](http://www.usecasemaps.org)



for filtering, and the way to analyze and represent the reduced traces so that they can express knowledge about the system.

Other dynamic analysis works employ web mining [58], association rules and clustering [35][34][45], and reduction techniques [16]<sup>6</sup>.

### 2.2.2.3 Module extraction

Hill Climbing and Simulated Annealing are used to form clusters of components based on fan-in/out analysis. The approach starts by building a Module Dependency Graph, then random partitions (clusters) of the graph are formed as the initial clustering configuration, and later the partitions are rearranged iteratively by changing one component from one cluster to another. The objective is to find the optimal configuration based on the concepts of low coupling and high cohesion, which is achieved by considering fan-in/out information. This was accomplished by maximizing the objective function, which the authors called Modularization Quality (MQ). The general process is shown in Figure 2.2.

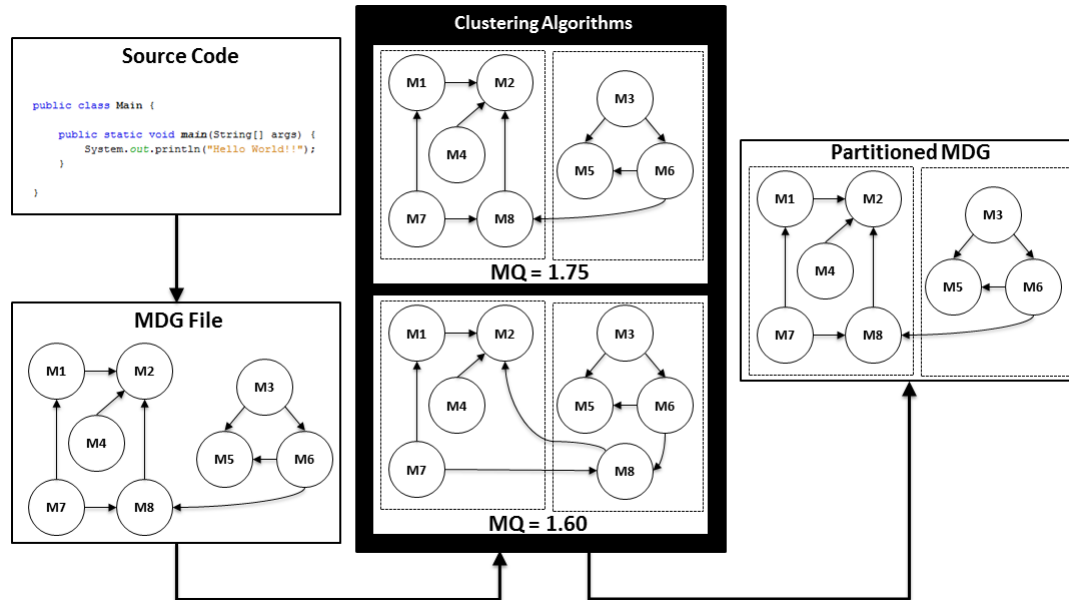


FIGURE 2.2. Modularization process based on clustering/searching algorithms [38].

### 2.2.2.4 Text processing

Text processing refers to the processing of source code as text. This implies the processing of morphological, syntactic, semantic and lexical elements of code. Text processing takes advantage of identifiers and how statements are organized to extract semantic information of artifacts: classes, methods, packages, etc. The requirement is that code is well written, i.e., the identifiers express semantic information of the business, and follow some general parameters about how they are defined.

In [42], the key-phrase extraction algorithm KEA<sup>7</sup> is used to translate business rules into specific domain business terms, from documentation. For this, they connect docu-

<sup>6</sup>For more information about dynamic analysis techniques see [15].

<sup>7</sup>For more information about the KEA algorithm see <http://www.nzdl.org/Kea> (November, 2012)

ments to business rules. The key point is that documents contain technical description of variables, so it is possible to establish a direct mapping between rules and documents.

## 2.3 Business Rules Extraction

*Business Rules Extraction* (BRE) is a reverse engineering process for recovering *Business Rules* (BR) from software information systems (Figure 2.3). In general, BR are constraints that define the structure of a business (i.e., structural business rules) and guide the way a business operates (i.e., operative business rules) [44][52]. BRE is important in software knowledge acquisition because:

- It is a means for software re-documentation [1] and functionality-code tracing [7].
- It builds BR-Code mappings that can support the understanding of the system [48].
- It supports the validation process for checking that the system fulfills its specification [7], i.e., that all the business rules are actually implemented [23].
- It is used in software re-engineering and migration [23].

In this section, the *Business Rule Extraction* related work is reviewed. This revision is the research background of the proposed approach for extracting structural BRs, described in the next chapter.

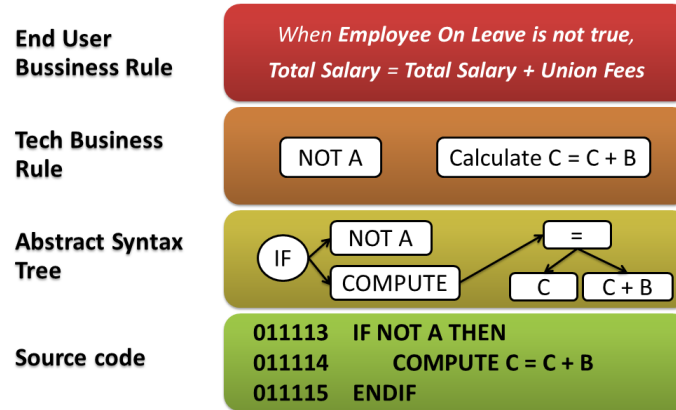


FIGURE 2.3. Business rules knowledge through abstraction levels.

We overview the BRE work in three categories: manual, heuristic, and dynamic. The former refers to conduct manual examination of source code for extracting BR, and the others about performing automatic extraction of rules. Heuristic techniques have focused on static processing of source code, while dynamic techniques emphasize the processing of dynamic artifacts, such as, execution traces. In general, automatic BRE has only focused on the examination of source code and on the identification of language structures or program sequences that could lead to business rules.

### 2.3.1 Manual BRE

Earls *et al.* [23] present a method for manual BRE on legacy code. The authors mention two BRE approaches: program-centric and data-centric. The method they propose is program-centric and focuses on locating and classifying error-processing sections in source code and the conditions that lead to those sections. The most important conclusion of this work is that manual BRE requires too much time, especially in large information systems, but is more accurate compared with existing and proposed automatic tools and methods.

In [54] the authors present an analysis on the impact of human factors in the BRE of legacy systems and survey additional related work to manual BRE.

### 2.3.2 Heuristic BRE

Heuristic BRE consists on automatic extraction techniques that take advantage of some common elements of the source code that could compose BRs. Elements considered in heuristic BRE are, for example, identifier names, exception raising/handling statements, and control flow structures. One method generally used in heuristic BRE is program slicing [56].

The work presented in [30] and [54] focus on identifying business/domain variables and the application of generalized program slicing for those variables. There are some important facts to consider from these works. First, the input and output variables are important because they belong to data flow interfaces in programs; second, control statements are essential for BRE since they guide the program's execution flow; and third, the BR representation defines mostly the BR components.

Shekar *et al.* [46] focus on enterprise knowledge discovery from legacy systems. The authors consider more “semantic” source code elements for performing knowledge discovery. Examples of the elements are: output messages, semantic relationships between variables and table columns, variable usages, assignment and control statements, and variables used in database queries.

In [7], the authors suggest to use error messages, program comments, functions, and control flow structures. The authors present three steps to extract BRs, based on SBVR standard: extraction of business vocabulary, creation of rules using vocabulary, and “activity interleaving”.

The authors in [41] perform BRE on COBOL legacy systems. They focus on simple COBOL statements that carry business meaning, such as, calculations and branching statements. In addition, they use the identifiers and conditions to extract the meaning and context of BR. The format used for BRs is *[conditions] [actions]*, in which actions are completed if conditions are satisfied.

In [48], the following four program elements that compose business rules are proposed: results, arguments, assignments, and conditions. Using program slicing, the authors track all the assignments of data results from calculations and capture the conditions that trigger the assignments. In this work, meaningful names of variables is mandatory.

The authors in [53] propose the use of information-flow relations between input/output variables, statements and expressions to identify domain variables. The approach they propose is mostly useful on procedural code.

While all these techniques relate to our BRE approach, in as much as they are static, automated techniques, none of them has the same input and output format as our implementation, hence we could not use any of them in our work.

### 2.3.3 Dynamic BRE

The work in [19] and [31] are examples of dynamic BRE. They present process mining approaches, which are intended to recover decision rules and control flow of systems from logs and execution traces. According to the authors, the main advantage of analyzing dynamic artifacts is that it is possible to detect participants, responsibilities, and concurrent activities in processes [31]. However, logs and other sources of information should comply with certain characteristics that make possible to perform process mining.

## 2.4 Summary

In this chapter we focused on the theoretical background and related work concerning reverse engineering and business rules extraction. In the first instance, the RE concepts were covered, including its definition, its process, and its relationship of software maintenance and understanding. Later the chapter presents a partial revision of standard and specialized RE techniques. Some of the standard techniques are implemented in the created reverse engineering tool we created. Chapter exposes the details about this. Finally, the BRE related work is reviewed as the research background for the proposed BRE technique, which is defined and described in detail in the next Chapter.

---

---

# Automatic Extraction of Structural Business Rules from Legacy Databases

---

---

As described before, ITC has no formal current technical and domain documentation in SIFI, including of the BRs involved in SIFI's business processes. The knowledge about the system has remained mainly within two groups of people: the technical group, who knows SIFI's architecture, and the business group, who knows the business processes supported by SIFI. The problem is that when people leave the company, a loss of knowledge about the system and the business occurs. To mitigate this problem the company has started a reverse engineering process in which BRE is one of the most relevant steps. This Chapter reports the first steps of this process: an approach for extracting structural BRs from legacy databases.

In order to define the BRE approach, we performed a revision of the BR concepts, their characteristics and categorization, based on the Semantics of Business Vocabulary and Business Rules (SBVR) standard [5][26]. Then, we analyzed the structural database (DB) components of SIFI and the BR concepts to define a mapping among them, using basic heuristics. Finally, we defined a BR format, following the SBVR methodology for the extraction of rules. Our technique extracted 870 BRs from the SIFI databases. Four employees of ITC analyzed 300 of the rules and found that 29% of recovered rules are correct structural business rules, 36% correspond to implementation rules, and 35% are incomplete or incorrect rules. We further analyzed with the four evaluators the incomplete/incorrect rules to identify ways to improve them. The recovery technique proves to be practical, while there is room for improvement, and it will be used as basis for the recovery of additional knowledge.

The chapter presents first background information on BRs, followed by the definition of our approach for BRE. The evaluation process and the results are presented and discussed in Chapter 5.

## 3.1 Business Rules

Business rule is a common term in business and software design. Intuitively, we consider BR as what the business and software does or operates. Although this conception is not wrong, it is quite inaccurate.

### 3.1.1 Definition of Business Rules

A rule is an explicit regulation or principle that governs the conduct or procedures within a particular area of activity, defining what is allowed. A business rule is a rule under business jurisdiction, that is, a rule that is enacted, revised and discontinued by the business [26][44]. For example, the “law” of gravity can affect a particular business, but it is not a business rule because the business cannot govern it; instead, the business may create rules for adaptation or compliance.

Business rules guide the behavior or action of a particular business and serve as criterion for making decisions, as they are used for judging or evaluating a behavior or action. They shape the business (i.e., the business structure) and constraint processes (i.e., the behavior of the business), to get the best for the business as a whole [29][44]. This means that business rules must be defined and managed in an appropriate way to guide the business to an optimal state.

### 3.1.2 Structure of Business Rules

OMG’s Semantics of Business Vocabulary and Business Rules (SBVR) standard [5][26] defines the key concepts of BRs:

- **Business vocabulary:** is the common vocabulary of a business, built on concepts, terms, and fact types.
- **Business rules:** they are statements/sentences based on fact types that guide the structure or operation of a business.
- **Semantic formulation:** is a way of structuring the meaning of rules through several logical formulations [26], e.g., logical operators (*and*, *or*, *if-then*, etc.), quantification states (*each*, *at least*, *at most*, etc.), and modal formulations (*It is obligatory* or *It is necessary*).
- **Notation:** is the language used to write and express BRs. The SBVR standard uses three reference notations, namely: *SBVR Structured English*, *RuleSpeak*, and *Object-Role Modeling*.

Business rules are composed of a structured business vocabulary [44], which provides them with meaning and consistency. *Structured business vocabulary* comprises the following elements [29][44]:

- **Noun concepts:** they are represented by terms of the business. They are elemental, often countable and non-procedural. For instance, *Currency*, *Country*, *Customer*, *Operational Cost*, etc.

- **Instances:** they are “examples” of noun concepts. Instances are always from the real world, not in a model. For example, *Euro*, *United States*, etc.
- **Fact types:** they are connections of concepts made by verbs or verb phrases. They give structure to the business vocabulary, recognize known facts and organize knowledge about results of processes. Common shapes or classes of fact types are categorizations (e.g., *Current Account is a category of Bank Account*), properties (e.g., *Bank Account has Balance*), compositions (e.g., *Bank is composed of Customers and Funds*) and classifications (e.g., *Euro is classified as Currency*). Fact types also can be categorized by arity, which is the number of noun concepts in the fact type.

The *structured business vocabulary* can be represented in many forms. Ontologies or UML class diagrams are examples of such forms; they can be used for representing and structuring the vocabulary and the knowledge about a business domain.

Through semantic formulations, business rules add a sense of obligation or necessity and remove degrees of freedom to the structured business vocabulary [44] (see Figure 3.1). For example, for the fact type *Bank Account has Account ID*, a business rule could be *A Bank Account must have only one Account ID*. In this case, the business rule has quantifiers (*A*, *only one*) and an operative modal keyword (*must*) that restricts the fact type. Another form to express the same business rule is: *It is necessary that a Bank Account has exactly one Account ID*. In this case, the rule has the structural prefix modal keyword *it is necessary that* and the quantifiers *a* and *exactly one*.

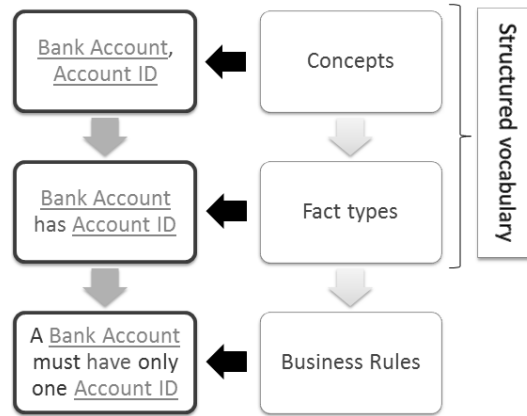


FIGURE 3.1. Business rules structure.

Although business rules can be expressed in several forms, it is required to follow only one specific notation, such as SBVR Structured English or RuleSpeak. Even so, and regardless the way business rules are expressed, they must be declarative and non-procedural, i.e., they must define the case/state of knowledge without describing when, where, or how the case/state is achieved [29][39][44].

### 3.1.3 Types of Business Rules

There are two types of business rules [5][6][44]:

- **Behavioral or operative rules:** they govern the behavior or business operations in a suitable and optimal fashion and, therefore, they are important for modeling

business processes. These rules always carry the sense of obligation or prohibition, can be violated directly [44], and not all are automatable. Examples of this kind of rules are the following:

- Not automatable: “*A Customer Service Operator must contact a Customer at least every month*”.
- Automatable: “*A Loan below \$1000 must be accepted without a Credit Check*”.
- **Definitional or structural rules:** they structure and organize basic business knowledge. They carry the sense of necessity or impossibility and cannot be violated directly. Unlike behavioral rules, not all definitional rules are business rules (e.g., law of gravity or rules of math); however, all of them are automatable. When evaluating structural rules, usually there are two possibilities: classifications (class membership) and computations (results). Some examples are:
  - Classification: “*A Customer is always a Premium Customer if the Customer has a Balance of more than \$1,000,000*”.
  - Computation: “*The Total Balance of a Bank Account is always computed as the Sum of Transaction Values*”.

### 3.1.4 What are not Business Rules?

Business rules, business policies, and advices are elements of guidance (guidelines) (see Figure 3.2). Then, what is the difference between them? why are all of them not considered as business rules? First of all, business rules must be practicable elements of guidance. Although business policies are guidelines, they are not practicable, so they are not business rules [44]. For example, the sentence “*Compliance to the Customer is our Priority*” is not a BR. Instead, business policies are reduced to practical guidelines, i.e., to business rules or advices. Advices and business rules are practicable guidelines because they are built upon fact types, but advices do not remove any degree of freedom from fact types [44]. In other words, they do not set any obligation or prohibition on business conduct, and any necessity or impossibility for knowledge about business operations [44]. The following is an example of an advice: “*A Customer Loan may be handled by a Personal Assistant*”.

In the same way, the following elements are not business rules:

- Events: they express actions performed by an actor in a specific moment in time. A business rule can be analyzed to find events where it needs to be evaluated.
- CRUD<sup>1</sup> operations: they are events that always result in data rather than a business rule.

Exceptions are not BRs per se, but violations to rules. However, they are used to formulate and organize logical and coherent BRs. In a business rule context there should not be exceptions; instead, well stated business rules.

---

<sup>1</sup>Create, Retrieve, Update and Delete.



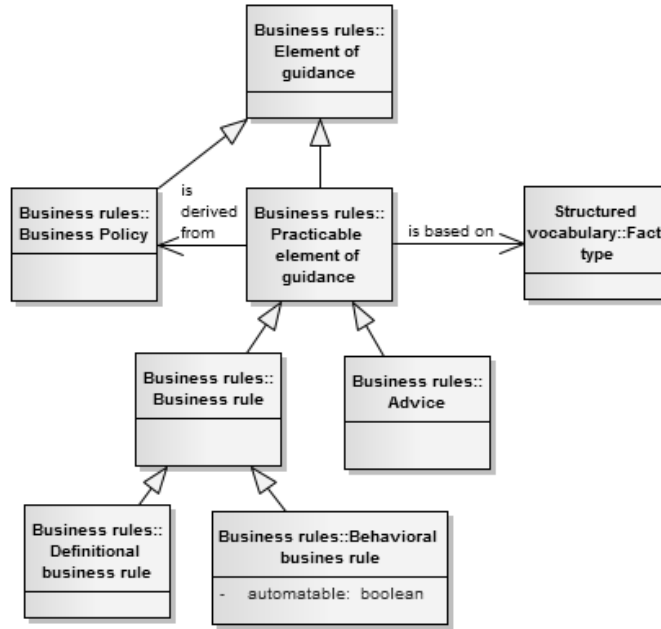


FIGURE 3.2. Element of guidance metamodel [44].

### 3.1.5 The Relation between Business Rules and Software Information Systems

Business rules capture the decision logic needed for activities in business processes and produce multiple events in processes [44]. Typically, software systems model and implement business process, so we can say that business logic and rules are in source code (see Figure 3.3), at least implicitly. For example, the system messages, when an operative exception has occurred, have implicit BRs, if they provide the users with guiding messages.

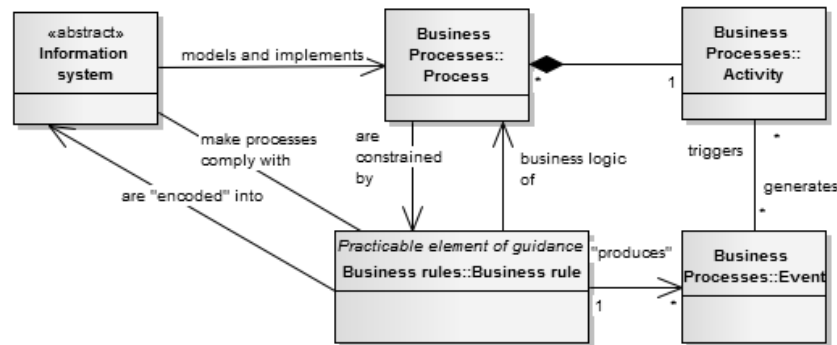


FIGURE 3.3. Relationship between information systems and business rules.

Business rules are key components within the structure of business processes and software models [6][44]. They evaluate facts in a process and can control the basis for changes in the flows of processes in which decisions appear. At business level, business rules enable the business to make consistent operative decisions, to coordinate processes, and to apply specialized know-how in the context of some product/service [44]. At software level, business rules guide flows in procedures and constraints the facts allowed. In any case, if a business rule changes, the business processes and the software modules associated to that

rule need to be changed [23], in order to restore the compliance with the business rules [7].

## 3.2 Extraction of Structural Business Rules

Business rules rely on fact types and, in turn, fact types on business vocabulary. For the automatic extraction of structural BRs from database components we followed the same reasoning. First, we extracted business concepts; then, we created fact types by relating the concepts through verb phrases and; finally, we generated sentences and business rules by adding quantifiers, modal and logical keywords to fact types. In this process, we tried to map every structural database component, including tables, columns, constraints and comments, to the business rules structures, i.e., concepts, verb phrases, fact types and rules. We defined a set of heuristics for BRE, which is used in an algorithm, and also a business rule DB model, which represents the format of the extracted rules.

For defining and refining the mappings and heuristics, we analyzed the database of SIFI. Because of the large size of the system, we decided to start the work on the *Programming and Payment* (PP) module, which is one of the largest modules of the system and it is used by almost all of the other SIFI modules. The module handles common tasks in many processes of a trust company: trust tax payments, contracts and invoice payments, investment discharges, and in general, payments to trust suppliers. We analyzed the portions of the SIFI database that correspond to the PP module.

### 3.2.1 DB - BR Mappings

We identified a set of database components to extract BRs from, and defined DB-BR mappings using basic heuristics (see Fig. 3.4). For this, we analyzed the DB core of the PP module, composed of 25 tables and considered standard DB elements, so that the heuristics would work for other systems. The following structural database components were considered<sup>2</sup>:

**Tables:** they often model noun concepts from the business domain. A table represents a unique concept, which is extracted from its comments and/or its name. For instance, the table called *FD-TMOVI* stores the movements of trust payments, so the concept of the table is *Trust Payment Movement*. Nevertheless, there are tables that only relate two or more tables (many-to-many relationships), resulting in those that represent verb phrases, instead of concepts. For example, the table called *FD-TMVCR* represents the relationship *Rejection Cause per Trust Payment Movement*. This table only relates the tables *FD-TCSRZ* (*Movement Rejection Cause*) and *FD-TMOVI*, thus representing a relationship. In this case, the table expresses the *has* relationship and the fact type *Trust Payment Movement has Rejection Cause*.

**Table columns (attributes):** they often model noun concepts and fact types. A table column also has a unique concept, and its relationship with the table that belong to, represent a fact type. The comments of columns are used for extracting concepts and verb phrases. Initially, the fact types that can be created with column concepts are those of class property. For example, the column *MOVI-ESTADO*, which belongs to the table

<sup>2</sup>All the examples in this and the next sections are from the PP module. The original data is in Spanish and we translate here some of the examples.

*FD\_TMOVI*, has the concept *State* (i.e., *Estado* in Spanish), so the property fact type extracted from the column and the table is *Trust Payment Movement has State*.

**Foreign keys/constraints:** these constraints represent relationships between table concepts, i.e., fact types. The verb phrases of these relationships are extracted from the comments of the tables/columns that the constraints reference. For example, the table *FD\_TMOVI* has a foreign key that references the table called *FD\_TOPER* (which has the concept *Trust Movement Operation*); therefore, the fact type that represents the foreign key is *Trust Payment Movement generates Trust Movement Operation*. In this case, the verb *generates* is extracted from the comments of the columns involved in the constraint.

**Primary and unique keys/constraints:** these constraints are used only to ensure uniqueness and identification of data. Therefore they do not map to any concept of BR.

**Check constraints:** they are conditions that always should be satisfied when adding and updating data in tables. This means they cannot be violated and, therefore, they often encode structural BR. We found three types of check constraints in the PP module: *non null* constraints, *check list* constraints, and *others*.

*Non null constraints* verify that columns always have non null values. A *non null constraint* operates only on one column of a table. For example, the table *FD\_TMOVI* has a constraint that checks that column *MOVLESTADO* has non null values (*MOVLESTADO IS NOT NULL*). A BR that can be created, from the constraint, is *A Trust Payment Movement always has a State*, which relies on the fact type *Trust Payment Movement has State* (created from the concepts of the table and column, and their relationship).

*Check list constraints* verify that column values always are in a finite list of values. We found three possibilities regarding the meaning of values: values meaning *classes of concepts*, *states of concepts*, or *operative parameters*. The meaning of the values is automatically determined using the comments of the columns involved in the constraint. For this, it is expected that the meaning of each constraint value is explicitly defined in the comments.

Regarding the **classes of concepts**, we used the heuristic that classes are often represented by nouns. In this case, the nouns corresponding to each value are used to create categorization fact types. For example, the table called *GE\_TFORMULA* (*Formula of Movement Concept*) has a column called *FORM\_CLASE* (*Formula Class*) and a check list constraint with the code

---

```
FORM_CLASE IN ( 'CD' , 'SD' )
```

---

The value *'CD'* corresponds to the concept *Discount Formula* and the value *'SD'* to the concept *Non-discount Formula*. Both concepts are extracted from the nouns in the comments and represent categories of the column concept. Then, the created categorization fact types are: *Discount Formula is a category of Formula Class*, and *Non-discount Formula is a category of Formula Class*.

Regarding the values that encode **states of concepts**, the heuristic used was: words corresponding to each value are verbs in past participle or, in some cases, adjectives. For instance, the column *MOVLESTADO* (table *FD\_TMOVI*) is used in the check constraint's code

---

```
MOVLESTADO IN ( 'A' , 'I' , 'T' , 'P' , 'X' )
```

---

where *'A'* is *authorized*, *'I'* is *inserted*, and *'X'* is *canceled*, etc. For this type of check list constraints, the values are used to build unary fact types, such as: *Trust Payment*

*Movement is authorized* or *Trust Payment Movement is canceled*.

Regarding values that mean **operative parameters**, we found that in the PP module they are used for guiding the operations and processes in the system. In other words, they do not represent structural knowledge. For example, the column *FORM\_IMP\_MUNIC* (table *GE\_TFORMULA*), with the concept *City Tax*, is used in the check constraint's code

---

```
FORMIMP.MUNIC IN ( 'S' , 'N' )
```

---

where there are two values: 'S' (Yes) and 'N' (No). The column comment indicates an operative condition that means if the formula should or should not consider *City Taxes*.

Other check constraints are those that involve any other logical conditions that always should be satisfied. For example, the column called *MOVI\_VLRMOV* (table *FD\_TMOVI*), which is the *Movement Value*, is used in the check constraint's code

---

```
MOVLVLRMOV >= 0
```

---

which means that the value cannot be negative.

**Table/column comments:** comments are descriptions in natural language about tables and columns in the database. They are used to extract noun concepts and fact types related to all the other DB components. When comments are not available in the database, concepts and fact types must be manually assigned or extracted from other sources, such as labels in the presentation layer of the information system.

In the case of SIFI, we were able to automatically identify the concepts of tables and columns from their comments (in most cases), using a Part-Of-Speech (POS) tagger - we used the TreeTagger<sup>3</sup>, which works for Spanish. By analyzing the DB comments of the PP module, we also realized that comments often contain more than one noun or composed nouns; thus, based on some informal tests we made about the completeness of the concept regarding the number of nouns, we decided to join up to three nouns to create a concept. Additionally, for creating fact types, on each column comment we expect to find verbs that associate concepts. When the column comments only contain nouns, the extracted verb is the one used for property fact types, i.e., the verb *has*. When the comments contain verbs, they are identified with the POS tagger and used to create fact types. For example, for the fact type *Trust Payment Movement generates Trust Movement Operation*, the verb *generates* is extracted from the comment of the column *MOVI\_OPER* (table *FD\_TMOVI*). Finally, the POS tagger was used to identify nouns, verbs in past participle and adjectives, for establishing the meaning of values in check list constraints.

### 3.2.2 BR Format and Schema

Based on *SBVR Structured English* and *RuleSpeak*, we defined a BR format, which was implemented in a database schema. The schema represents the basis for the development of an automated *General Rulebook System* (GRBS) [44], that allows to store and track all the knowledge around the BRs, including concepts, fact types, rules, and their relationships. The general goal of a GRBS is to provide means for the smart governance and a corporate memory through traceability [44]. Our schema has the same long term goal.

The schema we defined is composed of eight database tables (see Fig. 3.5). Seven of them model the BR concepts and the other one, the *bs\_t\_object\_concept* table, models

---

<sup>3</sup>TreeTagger can be found at <http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/>.

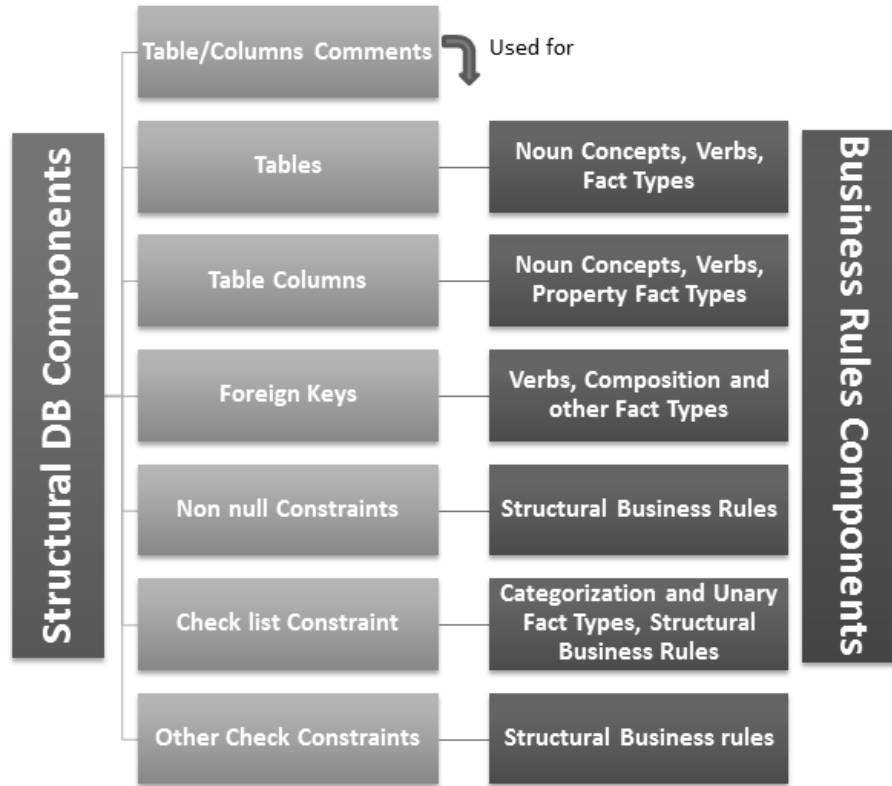


FIGURE 3.4. DB - BR component mappings.

the links between the DB components and noun concepts. As a result, the BR schema not only models and formats BRs but also provides support for tracking the relationships between DB and BR components.

According to the schema, concepts and verbs are terms used to compose fact types. In turn, fact types are complemented with quantifiers and keywords to compose sentences; and finally, the combination of sentences and keywords, compose BRs. Examples of quantifiers are the following: *every*, *at least*, *maximum*, *exactly*, *more than one*, and *between*. Keywords are logical (*if*, *only if*, *and*, *or*, etc.), operative (*must*, *can*) or structural (*always*, *never*).

A sentence is composed of two quantifiers, one per fact type concept, and a keyword. For instance, the sentence *A Trust Payment Movement always has a State* has the quantifier “*a*” for both concepts and the structural keyword “*always*”. In turn, a BR is a sentence or a composition of two sentences joined together with another keyword. For example, a BR composed of two sentences is: *A Trust Payment Movement only is authorized if the Movement has no Rejection Causes*.

There are two important design elements of the schema: a fact type can be composed by one or two concepts, having unary and binary fact types only; and in the same way, a business rule is exclusively composed of one or two sentences. These design decisions were based on the BR reduction principle expressed in [44]. The principle aims at producing easily-understood and granular rules that can be independently managed, re-used, and modified.

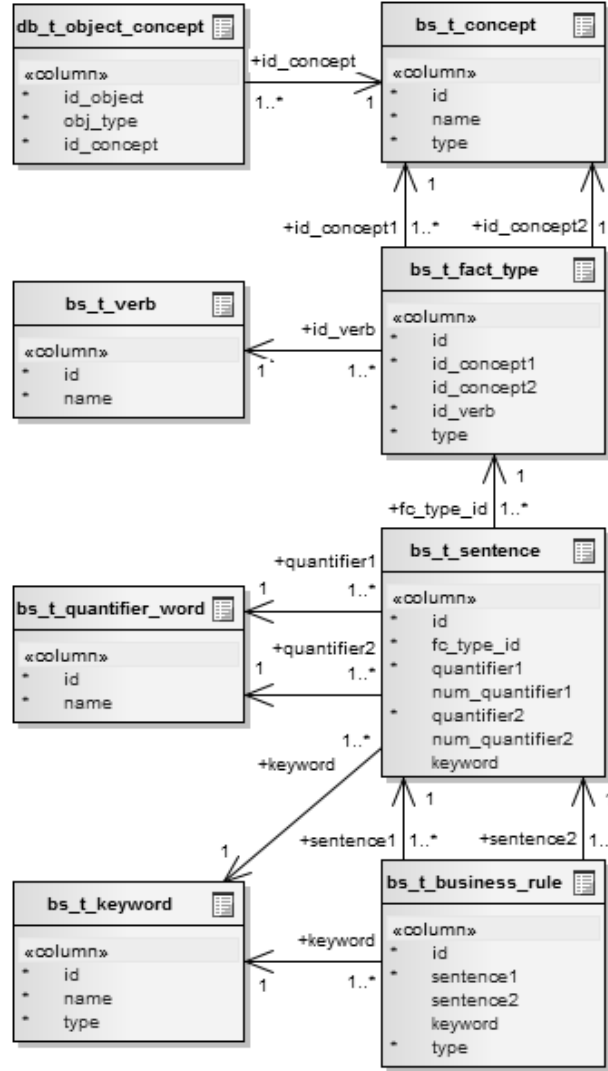


FIGURE 3.5. Implemented BR database model.

### 3.2.3 BRE Algorithm

The BRE algorithm has two parts: (1) extraction and processing of concepts and fact types; and (2) creation of the business rules. The input of the algorithm is a list of tables of the target system (in this case SIFI), which are processed one by one, and the output is a set of persisted concepts, fact types, and rules, following the schema defined above. From each table, the comments and the structural elements mentioned above are used.

The first part of the algorithm creates the structural vocabulary, necessary for generating BRs. The following steps are followed in the first part:

- For each table extract the comments. From the comments extract the nouns. Create the noun concept by using up to the first three nouns encountered in the comment. If there are no nouns or no comments, then use the table name as noun concept. Insert the noun concept in the database (unless already exists). The relationship between the noun concept and the table is inserted (for DB-BR component tracking).

- For each column that does not correspond to the foreign or primary keys extract the comments. If the comment contains the word “*if*”, then do not process it, else extract the nouns. Create the noun concept by using up to the first three nouns encountered in the comment. If there are no nouns or no comments, then use the column name as noun concept. Insert the noun concept in the database (unless already exists). The relationship between the noun concept and the column is inserted.
- For each check constraint in a column. Extract the nouns, the past participle verbs, and the adjectives from the comment of the column.  
If the check constraint is a list constraint (the PL/SQL condition of the constraint is parsed and verified), then extract the values of the list. If these values only include “*Yes/No*”, then exclude the constraint, because they do not represent structural constraints (instead, they are used for guiding procedures).  
If the number of nouns is greater than 2 and the number of past participle verbs is less than 2, then the values represent classes of the column concept, so insert each value as a new concept and create a category fact type between this new concept and the column concept. Else, the values represent states of the column concept, so insert each value as a verb phrase, thus creating unary fact types.
- Obtain the foreign keys for each table and its parent tables. For each foreign key, obtain the columns of the table involved in the foreign key. Identify the verbs from the comments of the columns. If there are no verbs then use the verb phrase “*has/belongs to*” as the verb for the fact type created between the table concept and the concepts of its parent tables. Else concatenate the verbs as the verb phrase of the fact type.  
Repeat the procedure with the children tables.

The second part of the algorithm generates the BRs:

- For each table, its parents and its children tables, do the following: for each column that only have a non null constraints obtain the fact types related with the column (see above). For each fact type, create and insert the sentence and the business rule by adding the keywords “*each*”, “*always*”, and/or “*one*” to the fact type (following the schema defined). For example, for the fact type *Deposit has Transaction Value*, the keywords are placed always in this way: ***Every Deposit always has one Transaction Value.***

The summary of the heuristics used in the algorithm is presented in Table 3.1.

### 3.2.4 PP Module Specific Heuristics

The above algorithm evolved in several iterations, where the results were investigated manually by the developer of the algorithm (not an expert in the domain, yet knowledgeable) and new heuristics were introduced based on these results. Most of these heuristics were introduced to make the approach more conservative. Despite that these heuristics could affect the generalization of the approach, we find them useful for increasing the precision.

For example, three conditions were necessary to detect and avoid columns that represent operative conditions and non-structural rules: (1) the presence of parametric values in the check list constraints; (2) the presence of the word “*if*” in the column comments;

TABLE 3.1. Summary of the heuristics used in the BRE algorithm.

Concepts are created from the first three nouns in the comments of tables/columns.
Concepts are created with the name of the table/column if it was not possible to identify the nouns in the comments.
Unary fact types are created from check list constraints which values encode state of concepts, if the column comments contain more past participle verbs than nouns.
Category fact types are created from check list constraints which values encode classes of concepts, if the column comments contain more nouns than past participle verbs.
Property fact types created from columns that do not belong to FKs or PKs, with the verb “ <i>has</i> ”.
Property fact types are created with all the verbs extracted from the comments of the columns that belong to FKs.
Property fact types are created with the verb phrase “ <i>has/belong to</i> ” from the columns involved in FK, if it was not possible to identify the verbs in the comments.
Business rules created are from property fact types of columns involved in non null constraints (those having the verb “ <i>has</i> ”), by adding the keywords “ <i>each</i> ”, “ <i>always</i> ”, and “ <i>one</i> ”.

and (3) the acceptance of null values in the columns. This allowed us to filter out extracted BRs from 1,010 to 870.

Another heuristic addresses the presence of the word “*which*” in the column comments. We noticed that columns having this word represent elements of business operations. In other words, they must be used to extract operative BRs (non-structural BRs). The same case happened with columns that had values meaning states; generally, they represent completed actions and, therefore, they can be used to create operative BR. In any case, those concepts and fact types were extracted, and we expect to use them for proposing a method for extracting operative rules as future work.

Apart from that, we found some problems related with the design of SIFI. An issue is that some categories between concepts that were detected manually from foreign keys, could not be extracted automatically. For example, the table called *FD\_TFIDE*, which represents the concept *Trust*, has a foreign key to the table *GE\_TCIAS*, which represents the concept *Company*. This foreign key represents a category of *Company*, resulting in the categorization fact type *Trust is a category of Company*. We did not find an automatic way for extracting these kind of fact types. Apart from that, we found other cases in which tables have the unique role of relating tables (many-to-many relationships), resulting in composition fact types. At this stage we did not extract those fact types.

Another fact we noticed was that some presumed nullable columns may not be nullable in reality. In this case, we assumed the constraints that check the column values are in higher layers of the information system, i.e., in presentation or application logic layers. Extracting BR relevant information from those layers is subject of future work.



### 3.3 Summary

In this Chapter we presented the BRE approach for legacy databases. We performed a revision of the BR concepts based on the SBVR standard, defined DB-BR component mappings, heuristics, and a business rule format, concluding with the design of a BRE algorithm, which was implemented in the reverse engineering tool, described in detail in the next Chapter.

---

---

# ReTool: An Oracle Forms Reverse Engineering Tool

---

---

ReTool is a static Reverse Engineering tool for Oracle Forms information systems. The general purposes of the tool are basically two:

- To synthesize all the technical and domain knowledge around the target systems<sup>1</sup>, and
- To support the maintenance and understanding processes in Oracle Forms systems.

ReTool also has a specific purpose, which consists on applying these general purposes to SIFI, the main information system that ITC develops and maintains, in order to tackle the maintenance issues of the system.

The general features of the tool are:

- Extraction, analysis and persistence of the structure and dependencies of forms and database objects.
- Simple and advanced searching of objects.
- Graphical browsing of dependencies between objects: object calls, queries and table references, and sequences of calls.
- Visualization of the structure and components of objects (forms, tables, packages, etc.)
- Source code visualization of executable PL/SQL objects.
- Data extraction for the target system database, through the generation of SQL INSERT statements.
- Data export of the processed information to Excel files.

---

<sup>1</sup>The target system is the software system subjected to reverse engineering (e.g., SIFI).

The development of the tool was led by the author of this thesis, iteratively and incrementally, receiving the feedback of the ITC's developers. The final product is a modular and useful tool, which effectively has supported them in their tasks since earlier versions of the tool.

This chapter describes ReTool in detail and is organized as follows: section 4.1 describes the Oracle Forms Technology (version 6i), emphasizing on the problems of this technology from the maintenance point of view, sections 4.2 and 4.3 present the tool in detail, including its architecture, its components and features. Some technical and design aspects of the tool are described in Section 4.4. Finally, the limitations and some general improvements to the tool are covered in Section 4.5.

## 4.1 Oracle Forms Technology

Oracle Forms version 6i is an enterprise application technology that provides a two-layer distributed client-server architecture [55]. The user interaction with the system is achieved through several form modules and the input and processed data is stored in an Oracle Database. Basically, a form module has several visual components (Figure 4.1 shows an example of a form module): input text fields, checkboxes, buttons, radio buttons, titles, labels and so on.

FIGURE 4.1. Example of a form module of an Oracle Forms application.

Each visual element has execution units that are triggered when events occur, for example, when the user passes the mouse over an element. These execution units are called triggers (or form triggers) and their behavior is analogous to the database triggers, those that are executed when there is an insert, update or delete on a database table. Figure 4.2 presents the general architecture of Oracle Forms. An Oracle Forms application is composed of one or more forms (form modules), which are the graphical user interface of the application. Each form module contains one or more data blocks (or just blocks), and each one includes one or more items. Blocks are created to perform CRUD operations on data sources, often tables, while items are used to represent the columns of sources

(tables). Items and blocks have events that are handled by triggers which communicate with other execution objects: program units in the form, procedures in libraries, and packages and procedures of the database. Please refer to [3], for more information on the elements that compose a form module.

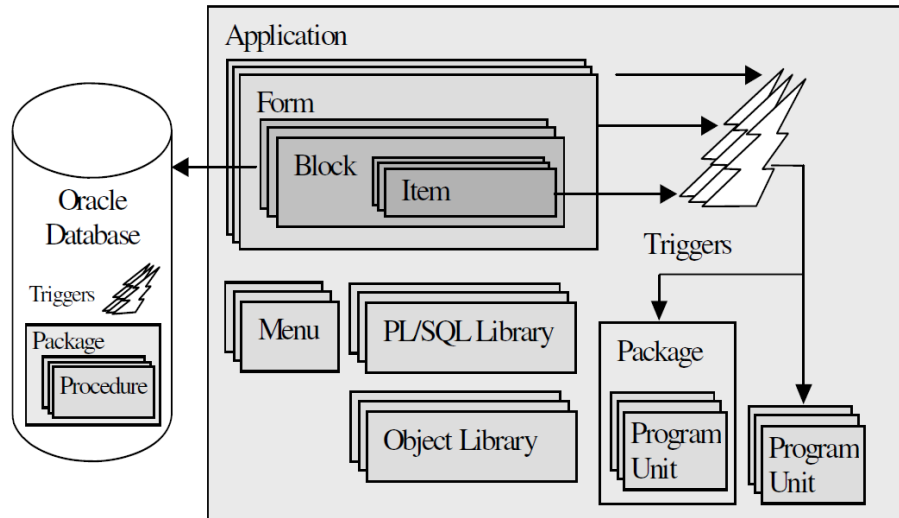


FIGURE 4.2. General architecture of an Oracle Forms application [3].

The business logic in Oracle Forms technology is implemented on both client and server through the PL/SQL language. This type of two-layer architectures has some advantages over one-layer ones [55], for example, there is a load distribution among clients, the job of the server is executed under only one context and the performance of the system is often high because of the high coupling possibility of the system. However, these advantages have some limitations from the evolutionary point of view [2][55]:

1. Scalability issues due to the centralized nature of the server and decentralization of the clients. A modification in the system often implies a large update in all the clients.
2. Strong coupling since the business logic can be integrated (mixed) with the view (the graphical user interface) and data management layers, and therefore, the business logic is difficult to isolate and change.
3. Strong dependence on the tools' vendor, which is risky if the vendor stops providing support. Also, these low-layer architectures have very poor portability.

Oracle Forms version 6i has been de-supported by Oracle since 2005. New versions of this technology include Web deployment, JEE and Web services integration<sup>2</sup>.

## 4.2 ReTool's Architecture

ReTool is organized in several macro components that are common in a reverse engineering tool [32]: extractors, analyzers, a repository and visualizers (Figure 4.3). The sources

<sup>2</sup>Source: [http://www.orafaq.com/wiki/Oracle\\_Forms](http://www.orafaq.com/wiki/Oracle_Forms)

of information of the tool are the form modules and the database objects of the target system. First, the sources are processed by the extractors (Forms Parser, Forms and DB Analyzers), then, their output data is stored in the repository (Metadata database), later, the analyzers process and synthesize the extracted data into information in the repository (Forms and DB Analyzers), and finally, this information is displayed in a visualizer (Information Displayer and Web Visualizer). The previous is the general process that a developer should follow to reverse engineer an Oracle Forms system with ReTool.

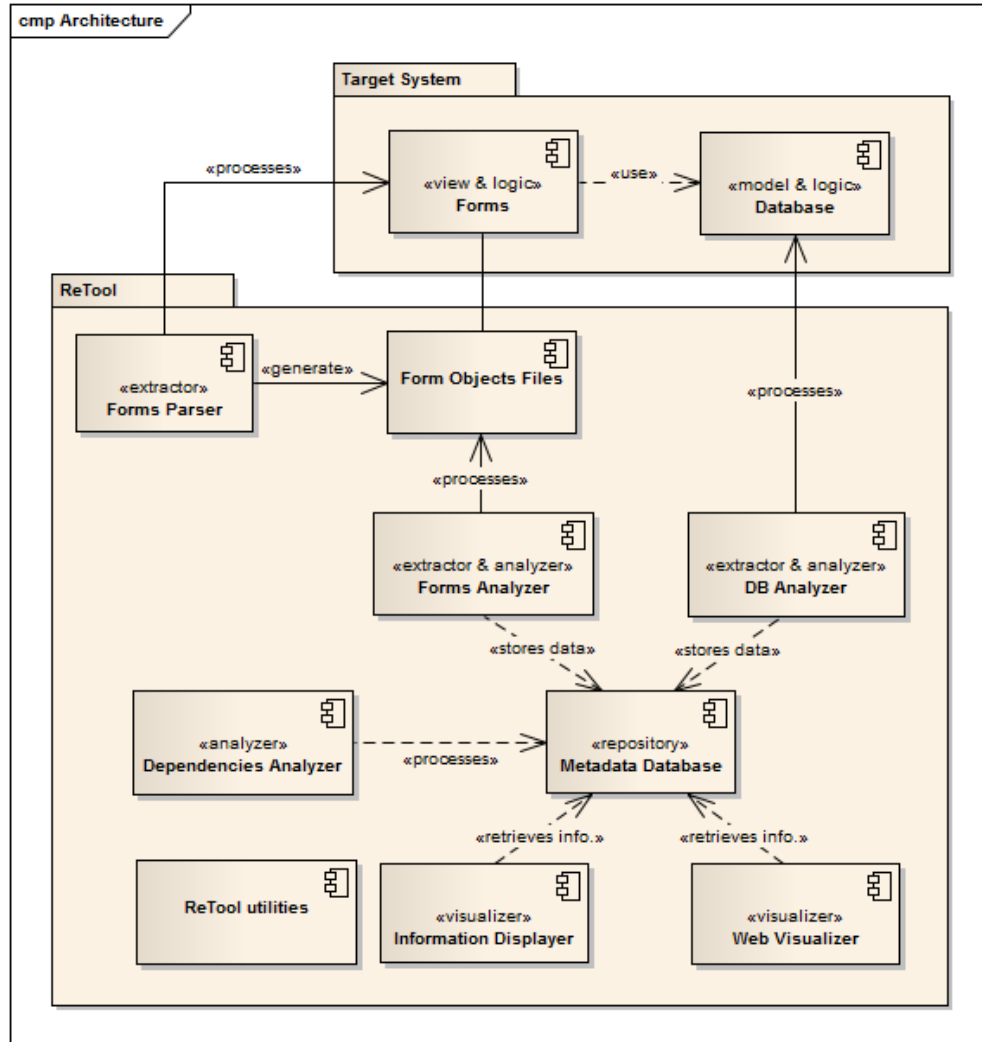


FIGURE 4.3. General architecture of ReTool.

### 4.3 ReTool's Components and features

This section describes each component of ReTool regarding its inputs/outputs, features and quality attributes, and the way it should be used.

#### 4.3.1 Extractors and Analyzers

ReTool has three extractors and analyzers:

**DB Analyzer:** this java command-line program processes the target system database, extracting all its structural objects, including tables, views, constraints, indexes and sequences, and executable objects such as PL/SQL executable packages, procedures, functions and triggers. This component also processes the calls between executable objects, at procedure level and also, the queries and tables that are referenced in them. All this information is stored in the ReTool repository. Figure 4.4 shows the command-line options of this program.

```
usage: db-analyzer [-a] [-b] [-cDbs] [-ct <objects>] [-dbs
<databaseSystem>] [-f <functions>] [-h] [-in <objects>] [-oc
<objects>] [-p <procedures>] [-pk <packages>] [-q <objects>] [-se
<objects>] [-t <tables>] [-tr <objects>] [-u] [-v] [-vw <views>]

Processes the database of a specified target system.

-a                processes all the objects
-b                processes the built-in objects
-cDbs             creates the target system if does not exist
-ct <objects>     processes the table constraints
-dbs <databaseSystem> name of the target system to process
-f <functions>    processes the functions
-h                print the messages of the analyzer
-in <objects>     processes the indexes
-oc <objects>     processes the calls of executable objects
-p <procedures>   processes the procedures
-pk <packages>    processes the packages
-q <objects>      processes the queries used by objects
-se <objects>     processes the sequences
-t <tables>       processes the tables
-tr <objects>     processes the triggers
-u                update the objects if they are already processed
-v                print the messages of the analyzer
-vw <views>       processes the views
```

FIGURE 4.4. Command-line options of the program DB Analyzer.

**Forms Parser:** written in C/C++, this command-line program processes the Oracle Form Source Code files (FMB files) of the form modules of the target system. It extracts the structure of a form and creates a hierarchical structure of folders and files with the properties of every element of a form, including data blocks, items, canvases, program units, triggers, alerts, list of values (LOVs), record groups and windows. Figure 4.5 shows, on the right, the structure of the form CUSTOMERS (visualized by the Oracle Form Builder, an Oracle development tool) and, on the left, the folder-files hierarchy that the Forms Parser creates when executed. Each element of the form has a XML file with all the properties of the element. Figure 4.6 shows an example of a generated XML file of the item COMMENTS. The parsing process from FMB files is possible using the Oracle Forms API [18].

Both analyzers, the DB and Forms Analyzers, use ANTLR Parser Generator v3 to parse and create the AST of procedures, functions and packages. A PL/SQL grammar was created based on the contribution by Patrick Higging<sup>3</sup>.

**Forms Analyzer:** written in Java, this command-line program processes all the folders and files generated by the Forms Parser. This program processes all the extracted components of a form, the calls of executable objects (program units and form triggers)

<sup>3</sup><http://www.antlr.org/grammar/1279318813752/PLSQL.g>

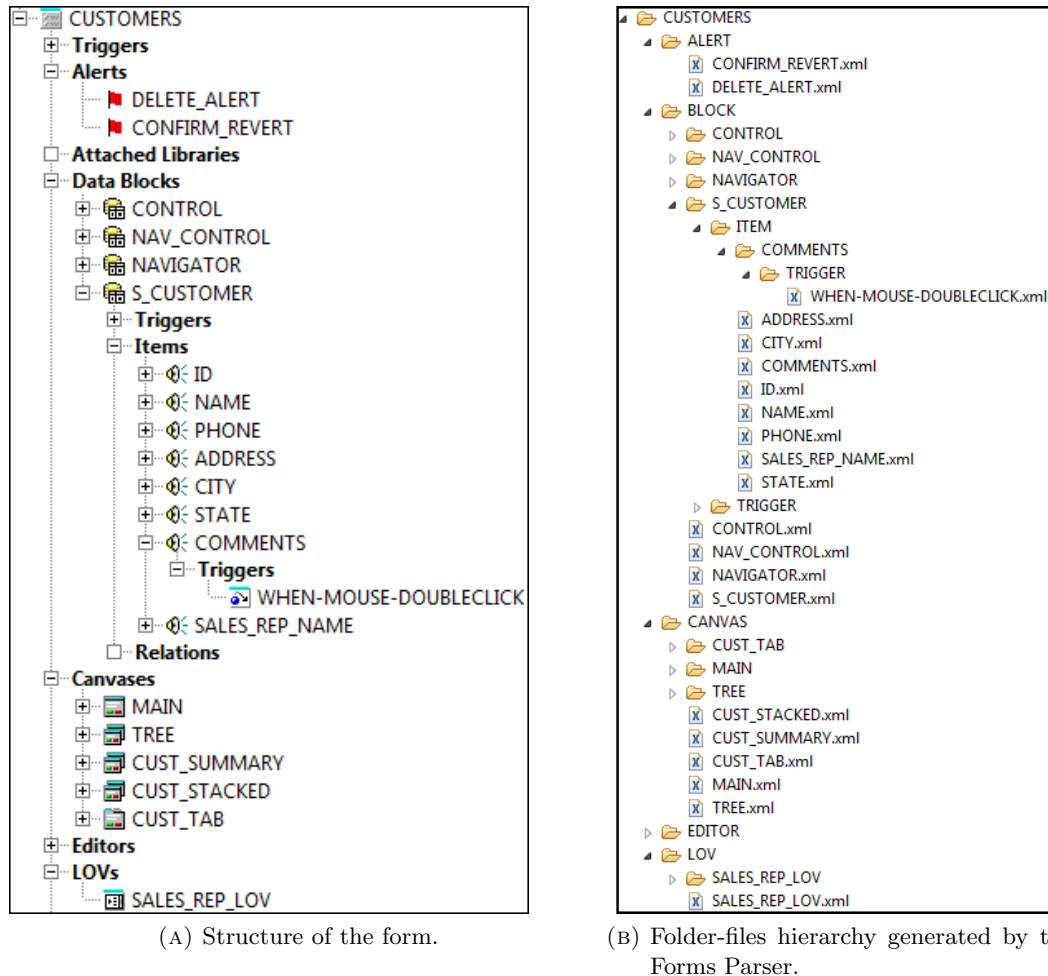


FIGURE 4.5. Comparison of the structure of the form CUSTOMERS and output of the Forms Parser.

and the queries and tables referenced by these objects. In the same way, the program is able to extract the referenced tables and views in data blocks of a form. Figure 4.7 shows the command-line options of the program.

**Dependencies Analyzer:** written in Java, this command-line program processes all the calls among the executable objects and the tables referenced by those objects, to build a dependencies tree. In this way, it is possible to know all the paths (sequences of calls/dependencies) among objects and 'directly or indirectly' all the tables that are referenced by a form and its components. The complete hierarchy of tables are also processed by this program and stored in the repository. Figure 4.8 shows the command-line options of the program.

### 4.3.2 Repository

The ReTool's repository is a metadata database, designed for the Oracle RDBMS, that stores every Oracle Forms/Database object and each of their attributes, for example, their name, the target system they belongs to, the source code in the cases of executable objects,

Database	
Database Item	Yes
Column Name	COMMENTS
Primary Key	No
Query Only	No
Insert Allowed	Yes
Update Allowed	Yes
Update Only if NULL	No
Physical	
Visible	Yes
Canvas	CUST_TAB
Tab Page	COMMENT
X Position	59
Y Position	14
Width	251
Height	79
Bevel	Lowered
Show Play Button	Yes
Show Record Button	No
Show Rewind Button	No
Show Fast Forward Button	No
Show Volume Control	Yes

(A) Properties of the element.

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<object name="COMMENTS" type="13" typename="ITEM" order="12">
  <property type="2" name="Audio Channels" propertynumber="16">0</property>
  <property type="1" name="Display Hint Automatically" propertynumber="19">0</property>
  <property type="1" name="Automatic Skip" propertynumber="23">0</property>
  <property type="2" name="Bevel" propertynumber="26">1</property>
  <property type="2" name="Calculation Mode" propertynumber="39">0</property>
  <property type="1" name="Case Insensitive Query" propertynumber="42">0</property>
  <property type="2" name="Case Restriction" propertynumber="43">0</property>
  <property type="2" name="Check Box Mapping of Other Values" propertynumber="47">1</property>
  <property type="2" name="Compression Quality" propertynumber="55">1</property>
  <property type="3" name="Canvas" propertynumber="57">CUST_TAB</property>
  <property type="3" name="Column Name" propertynumber="62">COMMENTS</property>
  <property type="2" name="Communication Mode" propertynumber="66">0</property>
  <property type="2" name="Compress" propertynumber="67">0</property>
  <property type="1" name="Conceal Data" propertynumber="70">0</property>
  <property type="2" name="Data Type" propertynumber="82">0</property>
  <property type="1" name="Database Item" propertynumber="84">1</property>
  <property type="1" name="Default Button" propertynumber="95">0</property>
  <property type="2" name="Display Quality" propertynumber="100">0</property>
  <property type="2" name="Distance Between Records" propertynumber="102">154</property>
  <property type="3" name="Editor" propertynumber="136">COMMENTS_EDITOR</property>
  <property type="2" name="Editor X Position" propertynumber="138">0</property>
  <property type="2" name="Editor Y Position" propertynumber="139">0</property>
  <property type="1" name="Enabled" propertynumber="140">1</property>
  <property type="2" name="Execution Mode" propertynumber="144">0</property>
  <property type="1" name="Fixed Length" propertynumber="149">0</property>
  <property type="2" name="Font Size" propertynumber="155">0</property>
  <property type="2" name="Font Spacing" propertynumber="156">4</property>
  <property type="2" name="Font Style" propertynumber="157">0</property>
  <property type="2" name="Font Weight" propertynumber="158">0</property>
  <property type="2" name="Height" propertynumber="190">79</property>
  <property type="1" name="Hide Object" propertynumber="193">0</property>
  <property type="1" name="Iconic" propertynumber="207">0</property>
  <property type="2" name="Image Depth" propertynumber="210">0</property>

```

(B) Generated XML file with information of the element.

FIGURE 4.6. Comparison of the elements of the item COMMENTS and the generated XML file by the Forms Parser.

the description or comment and their properties, in the case of Oracle Forms objects. The dependencies between objects are also stored in this database. The repository is divided in four parts:

- One portion stores the target system database objects and the dependencies between these objects. All these repository objects of this portion has the prefix “db\_”, as seen in Figure 4.9.
- Other portion includes the tables and objects related with target system forms. All these objects have the prefix “mm\_”.
- The tables and objects related with the parsing processes on the target system compose other repository portion. All these objects have the prefix “pr\_”.
- Finally, the rest includes the tables and objects related with the Business Process Extraction approach detailed in Chapter 3. All these objects have the prefix “bs\_”.

The database also has constraints that ensure the consistency and integrity of data and indexes that speed up the information retrieval of the visualizers. In addition, there are several database views that are useful for the visualizers as they condense and summarize information. Since it is expensive to build complete sequences of calls between two objects, as the amount of data could be very large, ReTool’s repository implements Materialized views [25] with indexes that speed up the queries. The dependencies tree generated by the Dependencies Analyzer is stored in two special tables and the materialized views are created referencing those tables. There are also tables and materialized views for the complete table hierarchy. The drawback with the materialized views approach is that they require to be refreshed in order to include new data in the repository. But the main advantages of the approach are the speeding of information retrieval and the possibility to know all the sequences of calls between objects.



```

usage: forms-analyzer [-a] [-al <alerts>] [-b] [-bl <blocks>] [-c
<canvases>] [-cDbs] [-dbs <databaseSystem>] [-f <folder>] [-h] [-i
<items>] [-l <lovs>] [-m <forms>] [-oc <objects>] [-p
<programUnits>] [-pm <popup menus>] [-pr] [-pri] [-q
<programUnits>] [-rg <rec groups>] [-t <triggers>] [-u] [-v] [-w
<windows>]

For a specified target system, processes the XML files and folders
generated by the Forms Parser.

-a                processes all the objects
-al <alerts>      processes the alerts
-b                processes the built-in objects
-bl <blocks>      processes the data blocks
-c <canvases>     processes the canvases
-cDbs             creates the target system if does not exist
-dbs <databaseSystem> name of the target system to process
-f <folder>       path of the folder where the XML files and
                  folders are stored
-h                print this help message
-i <items>        processes the items
-l <lovs>         processes the list of values (LOVs)
-m <forms>        processes the forms
-oc <objects>     processes the calls of executable objects
-p <programUnits> processes the program units
-pm <popup menus> processes the popup menus
-pr              processes all the properties of the objects,
                  except the items
-pri             processes the items' properties
-q <programUnits> processes the queries used by objects
-rg <rec groups>  processes the record groups
-t <triggers>     processes the form triggers
-u               update the objects if they are already
-v               print the messages of the analyzer
-w <windows>     processes the windows

```

FIGURE 4.7. Command-line options of the program Forms Analyzer.

```
usage: deps-analyzer [-a] [-cDbs] [-d <objects>] [-dbs <databaseSystem>]
                    [-h] [-m <forms>] [-pk <packages>] [-t <tables>] [-tm <forms>] [-tr
                    <triggers>] [-u] [-uv] [-v]
Processes the dependencies of a specified target system and generates the
dependencies tree.

-a                processes all the objects
-cDbs            creates the target system if does not exist
-d <objects>     processes the DB objects
-dbs <databaseSystem> name of the target system to process
-h              print this help message
-m <forms>       processes the forms
-pk <packages>   processes the DB packages
-t <tables>      processes the tables
-tm <forms>      updates the tables that are referenced by forms
-tr <triggers>   processes the triggers
-u              update the dependencies tree
-uv             update the materialized views
-v              print the messages of the analyzer
```

FIGURE 4.8. Command-line options of the program Dependencies Analyzer.

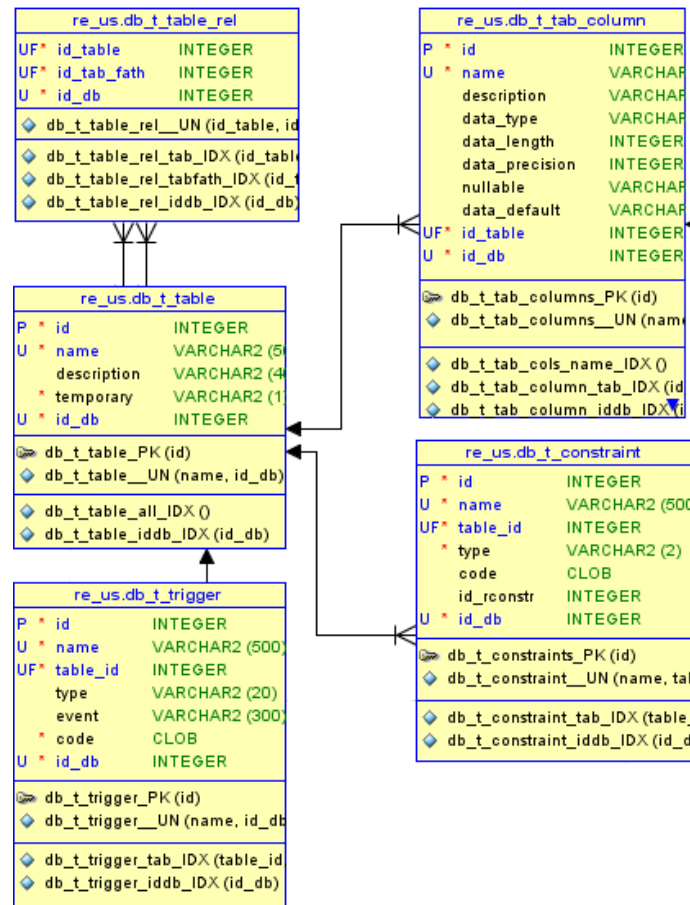


FIGURE 4.9. Example of the repository metamodel, which represents the database tables and their relational components, of the target system

### 4.3.3 Visualizers

The visualizers in ReTool display the information stored in the repository. There are two independent visualizers:

**Information Displayer:** written in Java, this command-line program receives a text string and the output is the information of the structure of the forms that contain the input text in their names. The program has several arguments that display or omit certain information; for example, there is an option for hiding the tables that program units and data blocks reference.

**Web Visualizer:** written in Java, this web application is the main visualizer and maybe the most important component of ReTool. Referred here as ReTool Web, it displays almost all the information stored in the repository under a uniform scheme of visualization. Most of the web pages of ReTool Web look like the one shown in Figure 4.10, in which there are several boxes that contain information, each box displaying the result of a fixed or dynamic query to the repository.

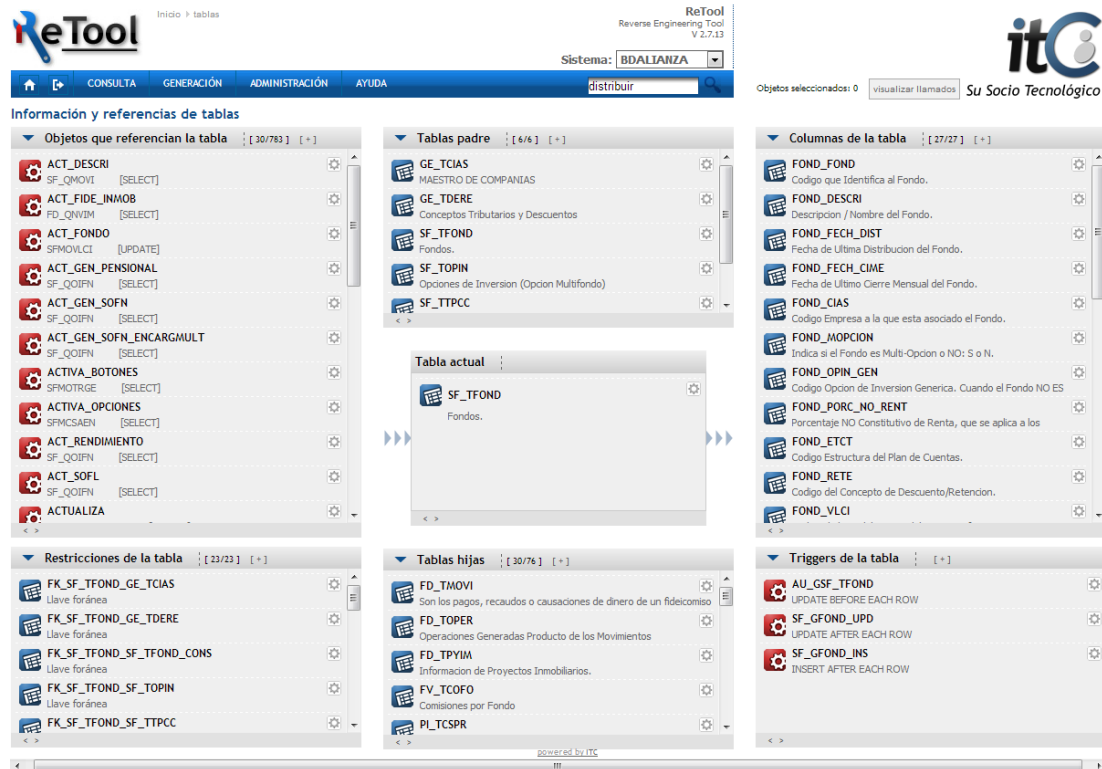


FIGURE 4.10. Graphical User Interface of ReTool Web.

As shown in Figure 4.11, a box of a web page is composed of some elements:

- A toolbar which contains, from left to right, a blue icon, the title of the box, the number of displayed objects versus the total of objects of the box and the graphical element [ + ].
  - The blue icon, if the mouse passes over it, displays a pop-up dialog box with some options that apply for the entire box. For example, in Figure 4.11, there

is a button in the pop-up box that exports all the data of the box to an Excel file (bottom-left corner image).

- The element [ + ] increases the size of the box to a fixed value, if the user requires to visualize more data in the box. The graphical element [ - ] allows to restore the box size.
- The content of the box include zero or more objects of the target system. Each one has an icon on the left, depending on the object type, its name, its description (under the name), and an icon on the right with several options that can be done on the object. The information presented in a box is retrieved in an on-demand and a paginated way. Each time the user scrolls down the box, the next page of the information is retrieved.

When the user clicks on the toolbar of a box, the content of the box is collapsed. If it is clicked again, then the box returns to its default size.

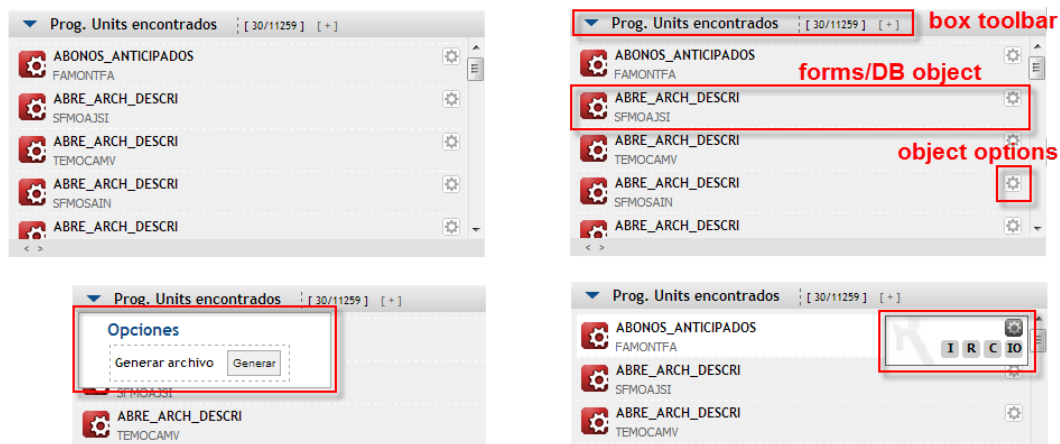


FIGURE 4.11. Graphical elements of a box.

Each option for the pop-up dialog of an object (bottom-right corner Figure 4.11) has a letter with a specific behavior on the object, if clicked:

- **I**: for all types of objects (forms, tables, packages, etc.), this option displays a pop-up box with specific information of the current object. For example, for table columns (Figure 4.12), the information displayed (from top to bottom) is the internal ID of the object, the object's name, its type, the type of data of the column, if it is nullable or not, and the table that it belongs to.
- **R**: for executable objects (except packages), tables and forms, this option redirects the user to the web page where the dependencies tree of the current object is being displayed.
- **In**: for tables, this option redirects the user to the page that generates the SQL INSERT scripts of the data stored in the target system database, based on a query on the current table.
- **J**: for tables, this option redirects the user to the table hierarchy page of the current table.

- **T**: for forms, this option redirects the user to the page where the tables referenced by the current form are being displayed.
- **F**: for tables, this option redirects the user to the page where the forms that reference the current table are being displayed.
- **P**: for all the components of a form, including data blocks, items, program units, and so on, this option displays the properties of the current component. This option is only displayed in the web page of the current form's structure.
- **C**: this option displays the source code of an executable object.
- **IO**: this option applies for executable objects (except packages), tables and forms. It adds the current object to the session, as a source or target object for the call sequences feature.



FIGURE 4.12. Pop-up dialog displayed for a table column FOND\_DESCRI.

The icons of the objects, according to the type of object, are shown in figure 4.13. The first three icons are the most used in the application, since the executable objects, forms and tables are the main and common objects in a Oracle Forms application (at least in SIFI). In the next paragraphs, every web page of ReTool Web is deeply described.

The first web page of ReTool Web is the login page (Figure 4.14). ReTool Web has a common authentication mechanism, in which a username and a password should be provided to access the application and the information it provides. Currently, there is only one user of the application: retool; more users will be provided and different access permissions to the application pages. Additionally, the login page has a selector of the target system in which the user is interested.

When the user has logged in, the searching page, shown in Figure 4.15, is presented to the user. All the pages of the visualizer have a header and body; the header has a target system selector and a menu bar with the following elements:

- The home and log out buttons.
- The menus and options for navigating to every page of the application.
- A simple searching input for quick searching. This allows the user to perform a simple search (on the objects name) in any page of the application.

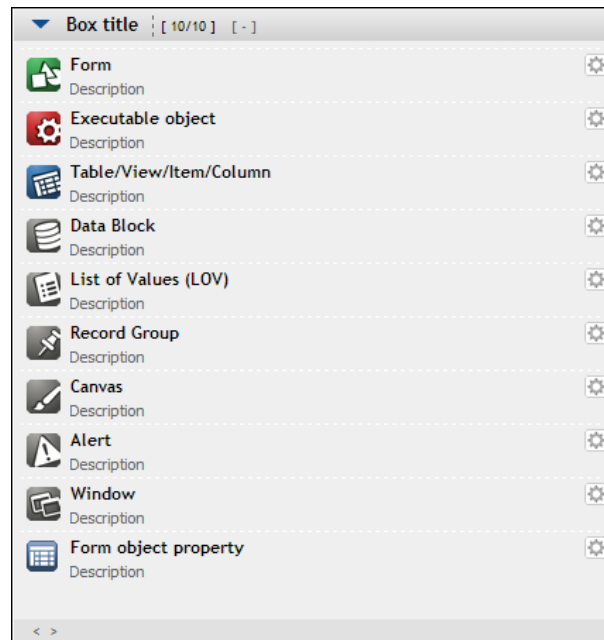


FIGURE 4.13. Icons of every type of object in ReTool Web.

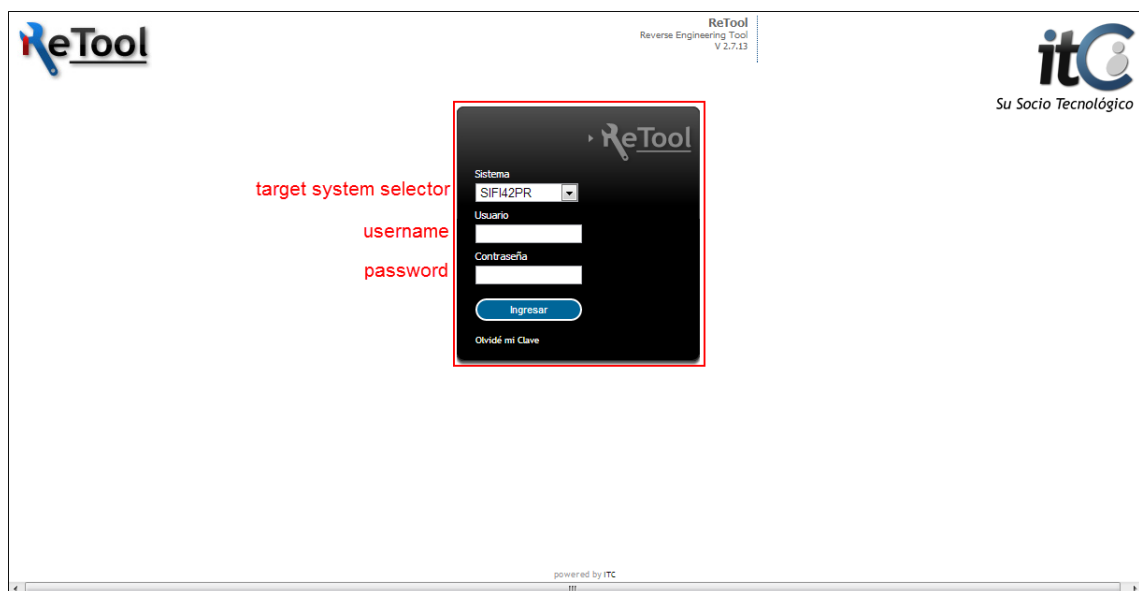


FIGURE 4.14. ReTool Web's login page.

In the center of the searching page, there is a searching input with the search and previous results buttons; on the right side of the page, there is a searching/filtering panel with several options such as, searching in the source code, in the comments of tables, or in the selected types of objects. Also, it is possible to search for objects that belong to other objects; for example, it is possible to search for columns of a set of tables by their names, or items that belong to a form specified by the user.

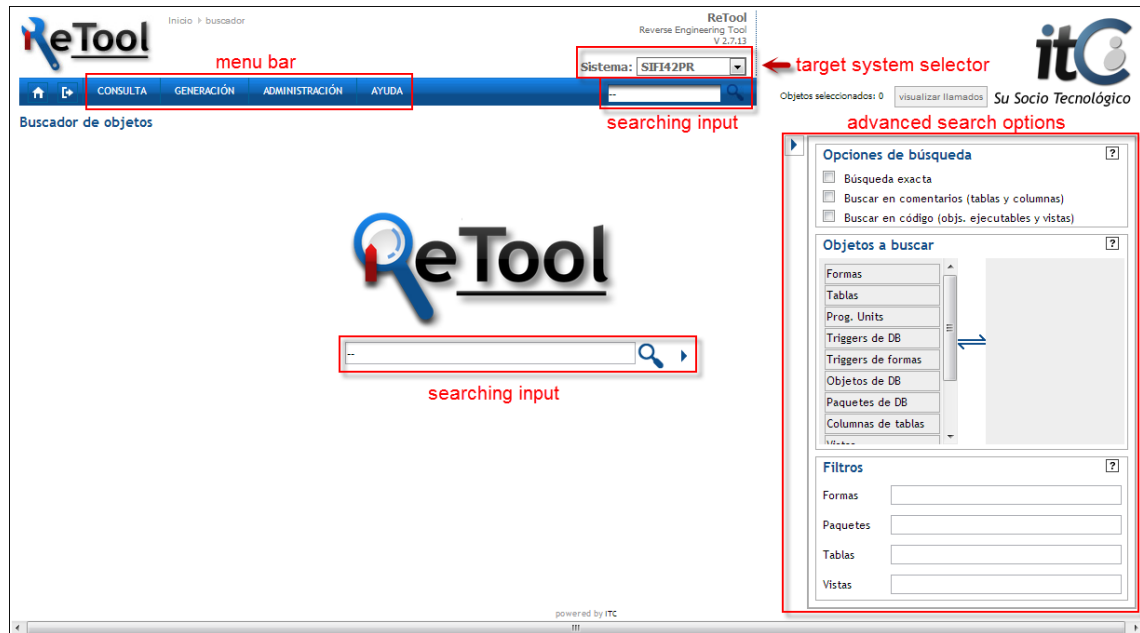


FIGURE 4.15. Searching page.

When the user performs a search, by entering a text string and pressing the searching button or the enter key, the results are shown as in Figure 4.16, where each type of object is located in a box. On the top-left of the results page, there is a button that takes the user to the searching form for performing another search.

When the user clicks on the name of an executable object, in the results page or any other page, the application redirects the user to the Object References page, show in Figure 4.17. This page has a special distribution:

- In the center of the page the Object Of Interest (OFI) is located.
- On the left column of the page, the boxes show the forms that reference the OFI and the tables that the OFI retrieves data from. In other words, the left column and the objects located there express some kind of “input” to the OFI.
- On the right column of the page, the tables that the OFI inserts, updates or deletes are shown. In this case, the column expresses some kind of “output” or result from the OFI.
- In the box above the OFI, the executable objects that call the OFI are displayed.
- Finally, the box below the OFI shows the executable objects that are called by the OFI.

ReTool Reverse Engineering Tool V 2.7.13

Sistema: SIF142PR

Objetos seleccionados: 0 visualizar llamados Su Socio Tecnológico

**Resultados de la búsqueda** - Sin opciones de búsqueda, Buscando en todos los objetos, Sin filtros

Nueva búsqueda

searching results by object type

Formas encontradas [30/1472] [+]	Prog. Units encontrados [40/14820] [+]	Tablas encontradas [30/3263] [+]
ADMCCLE ADMCCLE	ABONOS_ANTICIPADOS FAMONTFA	AD_TBIPG BITACORA DE PROGRAMAS
ADMCCONS ADMCCONS	ABRE_ARCH_DESCRI SFMOCGMF	AD_TBKTB TABLAS PARA COPIAS Y BORRADOS DE HISTORIA
ADMCEMPL ADMCEMPL	ABRE_ARCH_DESCRI SFMOCGMS	AD_TBKTR TABLAS RELACIONADAS O HIJAS PARA COPIAS Y BORRADOS DE
ADMCESCL ADMCESCL	ABRE_ARCH_DESCRI SFMODJISJ	AD_TCARG
ADMCRAC ADMCRAC	ABRE_ARCH_DESCRI	AD_TCLCN

Triggers de DB encontrados [30/614] [+]	Objetos de DB encontrados [30/6578] [+]	Triggers de formas encontrados [30/80541] [+]
AD_GCTVR_HIS INSERT OR UPDATE AFTER EACH ROW	ABONA FA_QCONTADO	COMENTARIOS TEMORINFO
AD_GCTVR_INS INSERT BEFORE EACH ROW	ABONA FA_QCLQOTA	KEY-CLRBLK SFMSPROF
AD_TVERSION_FTS_UPD UPDATE BEFORE EACH ROW	ABONA_DOCU_X_NOTAS FA_QNOTA	KEY-CLFRM SFMSPROF
AU_GFA_TMDTR INSERT OR UPDATE OR DELETE BEFORE EACH ROW	ABONA_LOTE FD_QPAGOS_INMOB	KEY-DELREC PIMOCODI
AU_GFD_TTPNV	ABONA_PAGVIIM	KEY-DELREC

Paquetes de DB encontrados [30/424] [+]	Vistas encontradas [30/741] [+]	Columnas de tablas encontradas [30/41859] [+]
AD_QAUDITORIA	CP_VCNFD Vista de Causaciones Mensualizadas	AAER_AUXI Numero del Auxiliar
AD_QCOMANDOS_SQL	CP_VORPA_IMPU Detalle de Impuestos.	AAER_AUXI_CLASE Tipo de Documento de Identidad
AD_QEXPO	CP_VTMPO Pagos de Órdenes de Pago (Formato1)	AAER_CODI Codigo de la entidad ante ante la Superbancaria
AD_QUTIL	CP_VTMPO2 Pagos de órdenes de pago (formato2)	AAER_FECH_CREA Fecha que Crea
AF_PAFDV2	FD_VBIEN	AAER_FECH_MODI

Bloques de formas encontrados [30/9863] [+]	Items de formas encontrados [30/122320] [+]	Columnas de vistas encontradas [30/11265] [+]
ABONOS FDMCSAFD	AAER_AUXI SCHSSUPE.SB_TAAER	ACCL_ACCION
ACRE SCHMUACRE	AAER_AUXI_CLASE SCHSSUPE.SB_TAAER	ACCL_CODIGO
ACRE SCHMUACRM	AAER_CODI SCHSSUPE.SB_TAAER	ACCN_ACCIONES
ACREEDOR FDMOORFD	AAER_FECH_CREA SCHSSUPE.SB_TAAER	ACCN_ACCN
ACREEDOR	AAER_FECH_MODI	ACCN_AUXI_EMIS

powered by ITC

FIGURE 4.16. Searching results page.



In this page, if the user clicks another executable object, the boxes are updated with the information of the new current OFI.

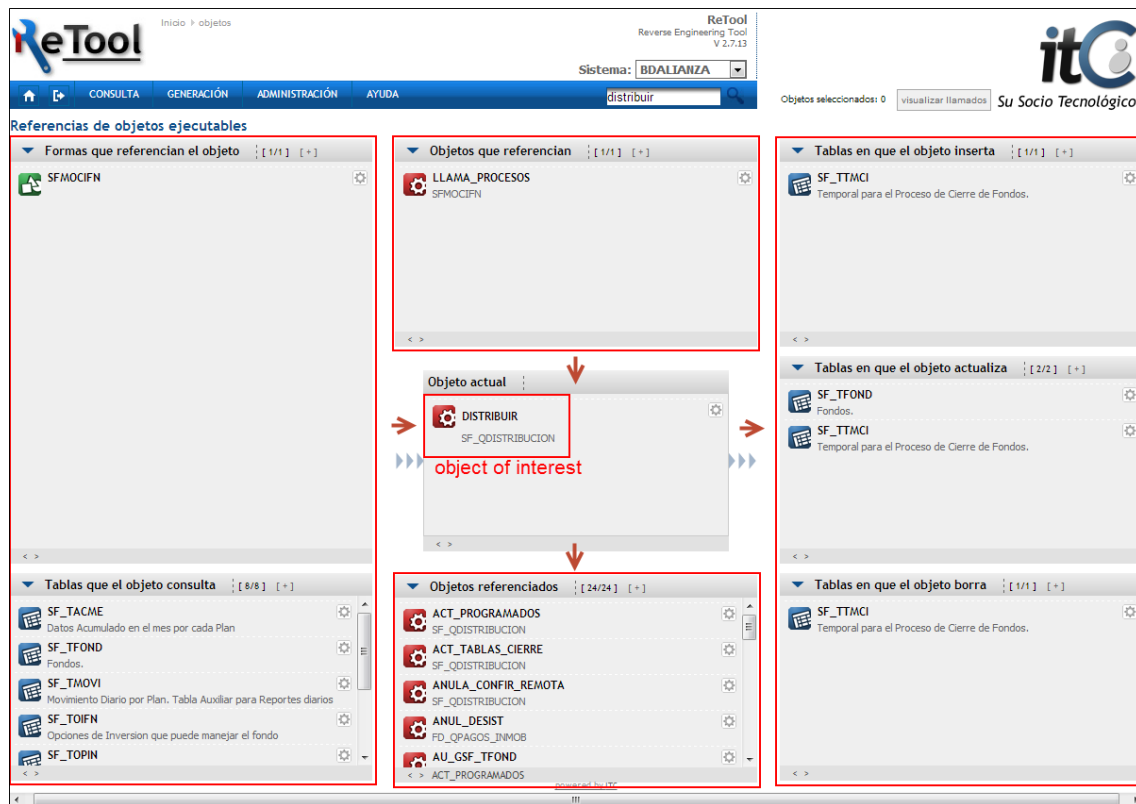


FIGURE 4.17. Object References page.

When the user clicks on the name of a table, the application redirects the user to the Table References page shown in Figure 4.18, which is very similar to the Object References page. The Table Of Interest (TOI) is located in the center, the top-left box shows the executable objects that perform any CRUD operation on the table, and the bottom-left box displays the table constraints. On the right, the table's columns and triggers are shown, while the box above the TOI contains the parent tables of the TOI and the box below the children tables <sup>4</sup>.

Now, when a form is clicked, the application redirects the user to the Form References page (Figure 4.19). The structure of the Form Of Interest (FOI) is shown on the left box. When the user expands a node of the tree, the objects are displayed in the table and also in the top-center box. The DB executable objects that the form references directly or indirectly are displayed on the bottom-center box. Finally, the properties of the current form object are shown in the right box. The current form object could be any object of the FOI, for example a specific data block or item that belongs to the FOI. For this, the users have to click the “proper” link of the object on the structure tree, or the **P** button of the object in the top-center box.

The Table Hierarchy page (Figure 4.20) presents the tree of all children tables of the TOI in the left box, and the tree of all the TOI's parent tables in the right box. To access

<sup>4</sup>A parent of a table is another table that has an integrity constraint (a foreign key) to the former one. The children table depends on the parent regarding the data it contains.

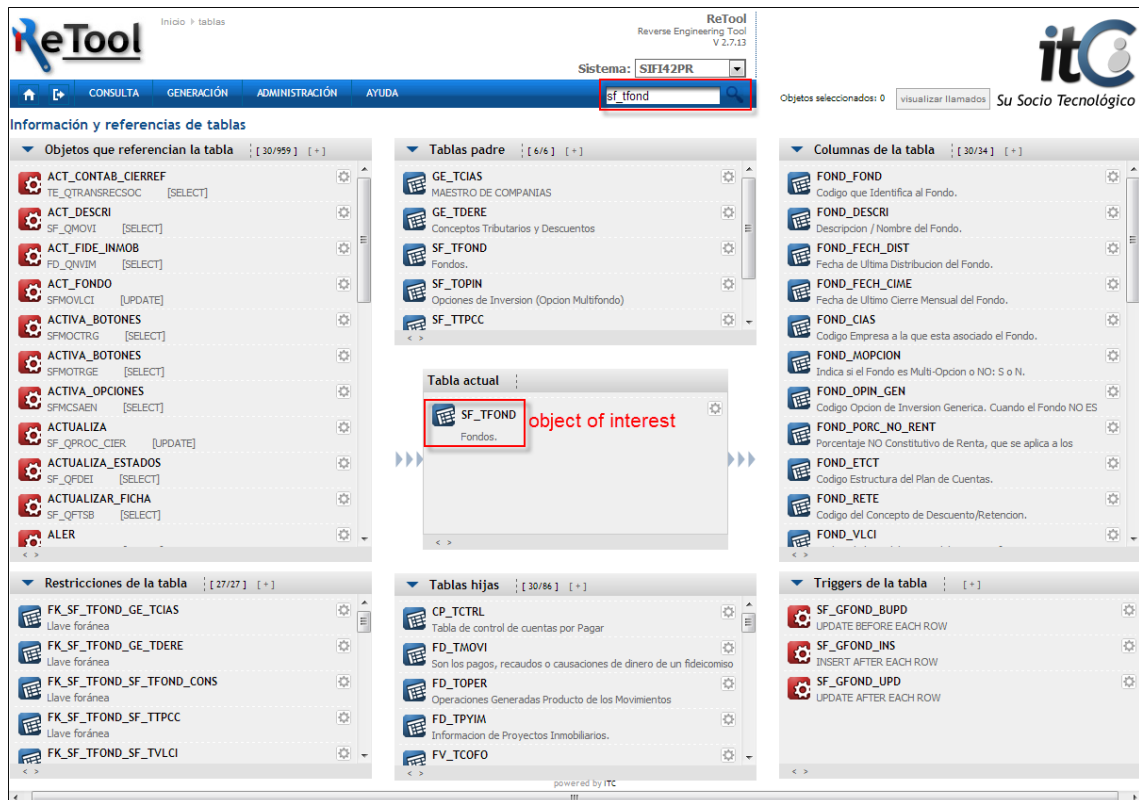


FIGURE 4.18. Table References page.

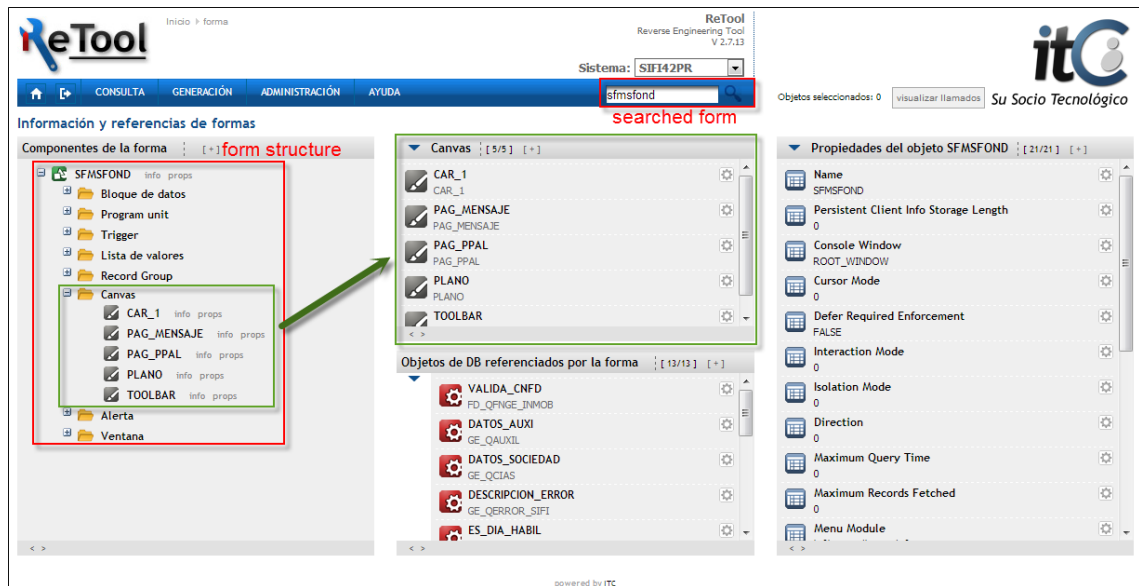


FIGURE 4.19. Form References page.

this page, the users need to click the **J** option of a table, in the pop-up dialog, as explained above.

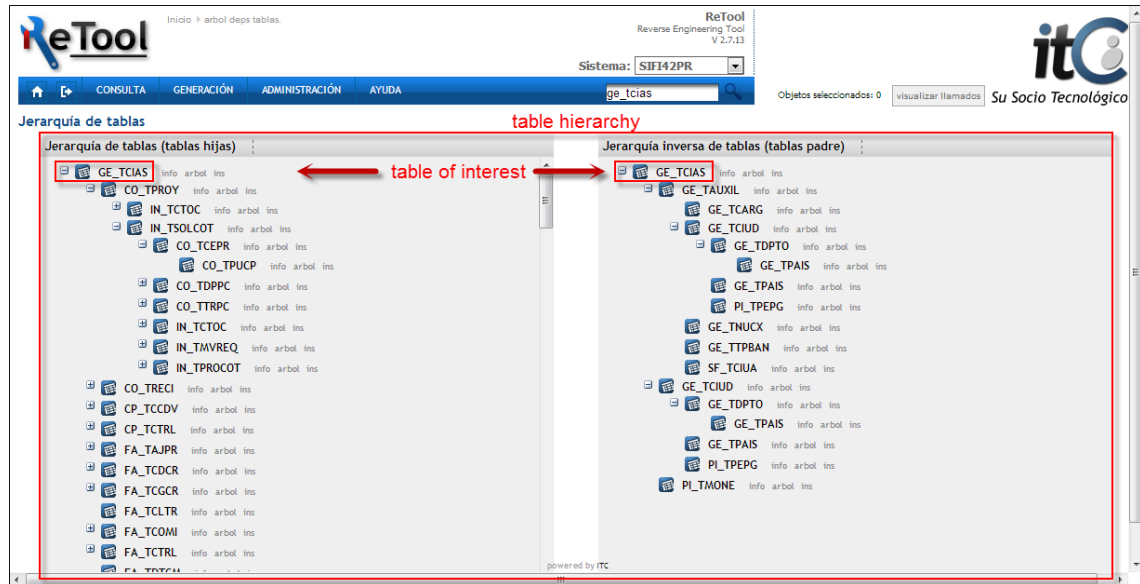


FIGURE 4.20. Table Hierarchy page.

One of the most important pages in ReTool Web is the Dependencies Tree page (Figure 4.21). Given a form, executable object or table, the direct or indirect forward dependencies are shown in the left box, and the reverse dependencies in the right box. In this way, it is possible to know sequences of calls and references between objects. For example, it is possible to navigate the references of a form and the references of those references until the referenced tables. The navigation in the opposite direction is also possible.

ReTool Web is also able to retrieve and display the sequences of calls between a source object and a target object (Figure 4.22). For this, in every page of the application, the user can select the source and target objects (only forms, executable objects or tables) with the **IO** option in the pop-up dialog of the objects. The selected objects are shown in the top-left corner of every page and they can be changed for other objects. When the source and target are already selected, the application enables the button that redirects to the Call Sequences page, above the source/target dialog. In this page, the source and the target are located in opposite directions, on the left and on the right, respectively. In the middle, all the sequences between them are displayed. Each sequence is in a frame in the middle box.

The application can also display information about a package (Figure 4.23). The Package Information page has three boxes: the left box displays the forms that reference the package, the middle box the executable objects (procedures and functions) that belong to the package, and the right box the tables that are referenced by the package. This page can be accessed by clicking a package.

The Tables-Form page (Figure 4.24) displays the tables that a form retrieves data (top-left box), updates (bottom-left box), inserts (top-right box), and deletes from (bottom-right box). The box below the FOI displays the tables that are data sources of the FOI's data blocks. This page is accessed by clicking the **T** option of a form in its pop-up dialog.

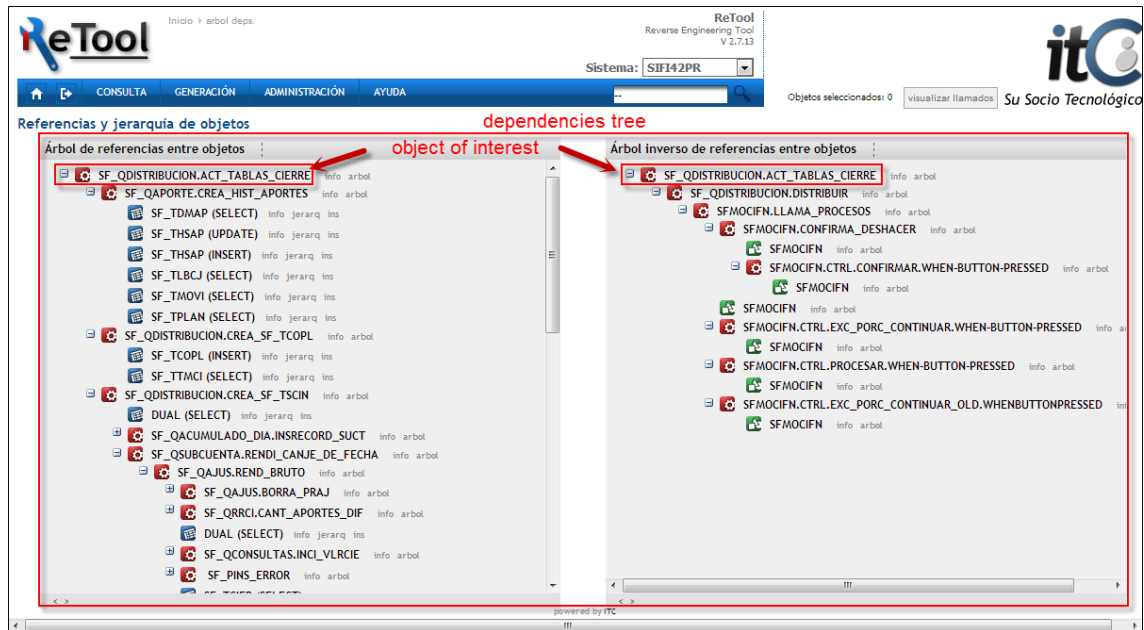


FIGURE 4.21. Dependencies Tree page.

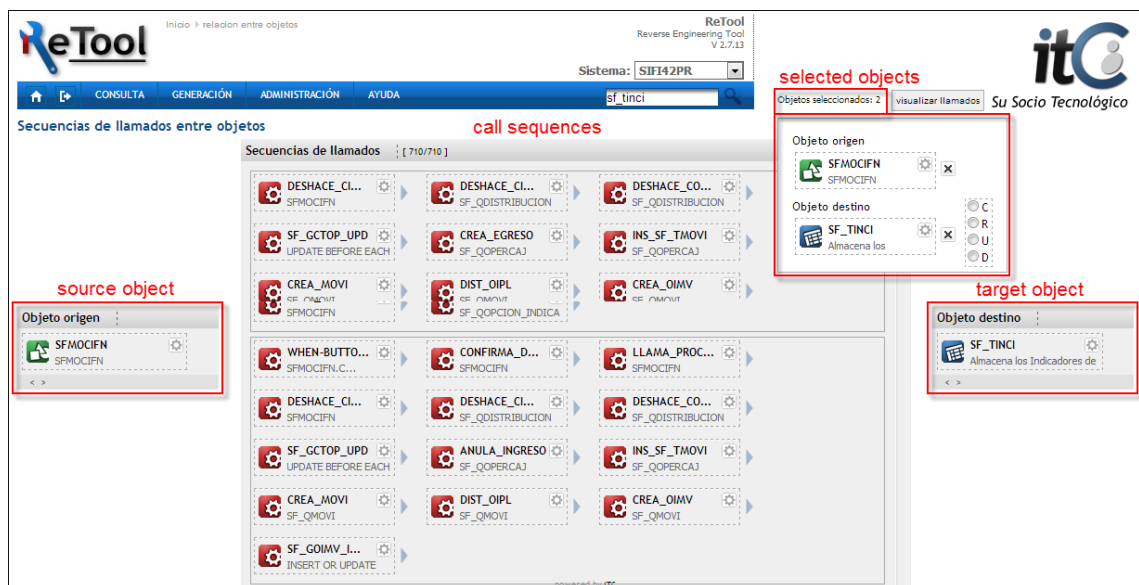


FIGURE 4.22. Call Sequences page.

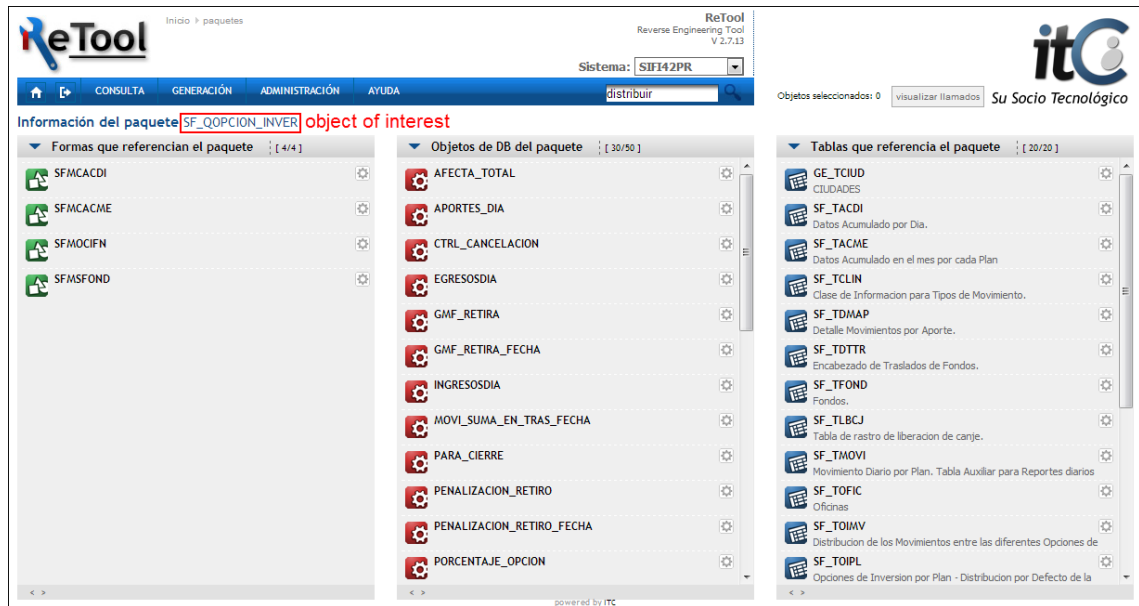


FIGURE 4.23. Package Information page.

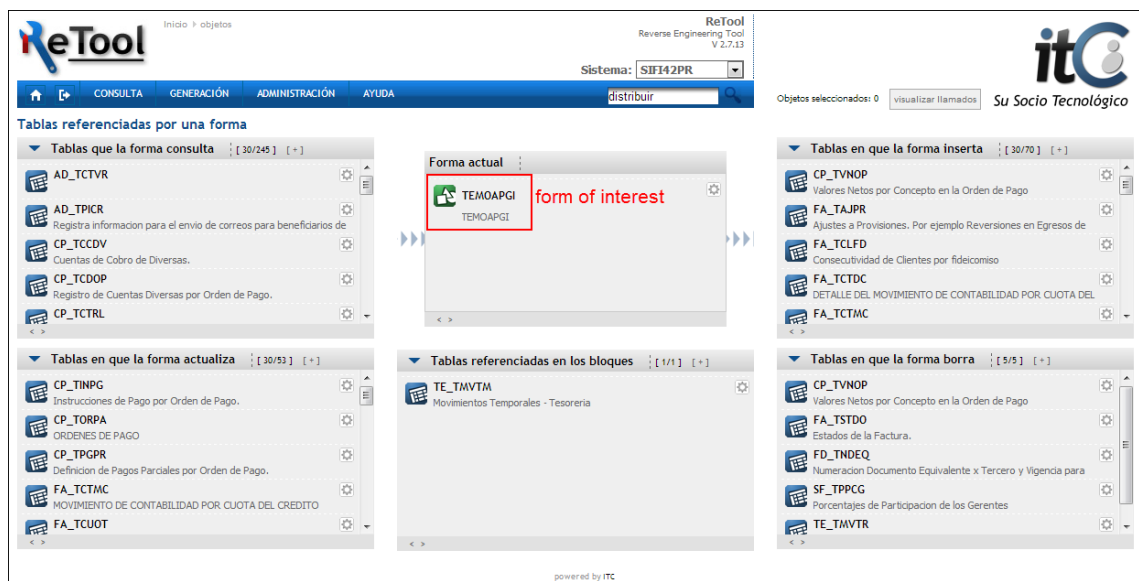


FIGURE 4.24. Tables-Form page.

In the same way, the Forms-Table page (Figure 4.25) displays the forms that insert (top-left box), delete (bottom-left box), retrieve data from (top-right box), and update (bottom-right box) the TOI. The box below the TOI displays the forms containing data blocks with the TOI as data source. This page is accessed by clicking the **F** option of a table in its pop-up dialog.

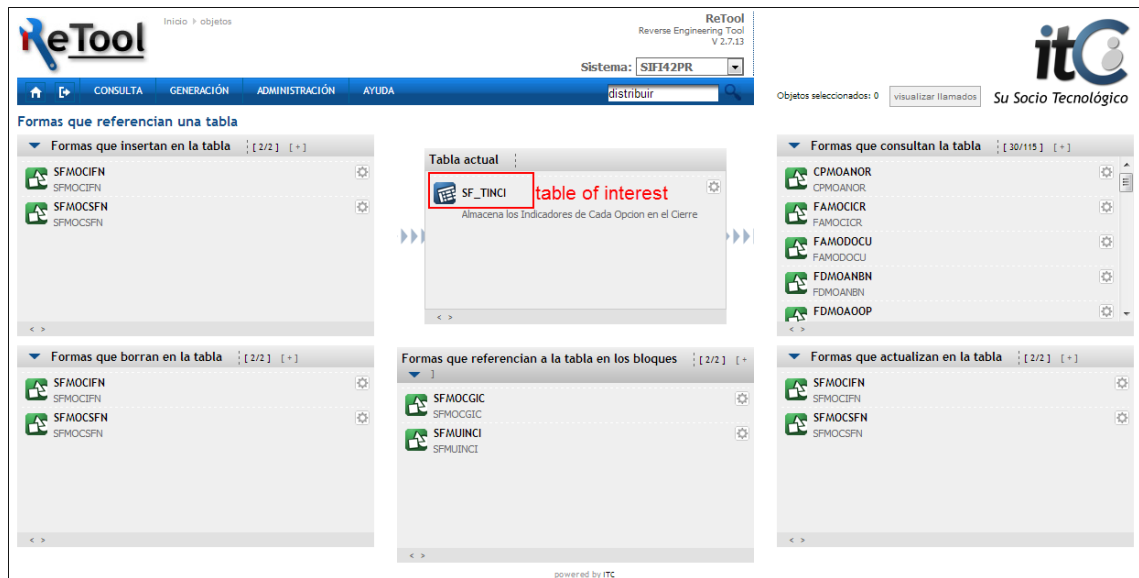


FIGURE 4.25. Forms-Table page.

In addition, ReTool is able to generate SQL INSERT statements of the data stored in the target system database (Figure 4.26). The SQL Inserts Generation page presents the TOI's columns on the left to the user, each one with an input box, and the user, after entering a searching value on one column, presses one of the three buttons (above the columns) for retrieving the data related with the TOI:

- If the user presses the left button, “parent inserts”, the application retrieves the data of the TOI and all its parent tables, the parents of its parents, and so on, and generates a script with the SQL INSERT statements of all the data, preserving the integrity order in which the statements should be executed. In this way, it is possible to use the generated script to create the data in an empty database.
- If the user presses the middle button “all inserts”, the application retrieves the data of the TOI, its parent and children tables. The application considers the cases when one child table has one or more parent tables different than the current table, so it is possible to retrieve the entire database (all the data). Thus, this feature should be used carefully, because retrieving all the data could take too much time and could also increase the load of the target database.
- If the users press the right button “all + triggers”, then the application does the same as the middle button, but at the beginning of the script, the statements that disable the triggers of the tables involved are placed and at the end of the script, the triggers are enabled again with the proper statements. This prevents some errors when the SQL INSERTs are executed.



[illegible]

FIGURE 4.26. SQL Inserts Generation page.

Finally, ReTool Web provides a web page that shows the information of the reverse engineering/parsing processes performed on the target system (Figure 4.27). On the left side of the page, all the processes are shown in decreasing order in time, the first process that appears is the last process performed on the system. On the right side, the general and the detailed information of the current selected process (by the user) is shown. The general information displayed includes the process state (*in progress*, *successfully finished*, and *finished with errors*), the type of processing (*DB*, *Forms*, *Dependencies*), the date and time when it started, the ending date-time, the spent time and the number of errors/warnings. The detailed information includes, for every type of object, the current number of objects processed versus the total number of objects to process, the progress of the processing, its state and the number of errors/warnings. This page is accessed only in the Administration menu in the menu bar.

In short, ReTool Web is a simple and versatile visualizer, which includes several pages that describe the structure of objects such as forms, tables and packages. One of the main general features of the tool is the object dependencies processing of executable objects, forms and tables, which can be browsed in several ways: in general, different web pages present this information, and in particular, the Dependencies Tree and the Call Sequences pages allow to explore all the direct and indirect dependencies of objects.

#### 4.3.4 Utilities

Additional to the core components, ReTool has several command-line programs that complete the component portfolio of the tool:

ReTool Reverse Engineering Tool V 2.7.13

Sistema: SIFI42PR

Inicio > procesamiento

CONSULTA GENERACIÓN ADMINISTRACIÓN AYUDA

ge\_tciud

Objetos seleccionados: 1 Visualizar llamados Su Socio Tecnológico

**Historial de Procesamiento** **target system**

**Procesos de parsing** **parsing processes**

(943) Dependencies parsing  
(934) Dependencies parsing  
(926) Database parsing  
(925) Dependencies parsing  
(924) Forms Parsing  
(923) Database parsing  
(899) Dependencies parsing  
(889) Dependencies parsing  
(888) Forms Parsing  
(887) Database parsing  
(870) Dependencies parsing  
(856) Dependencies parsing  
(855) Forms Parsing  
(854) Database parsing  
(832) Dependencies parsing  
(824) Dependencies parsing  
(822) Forms Parsing  
(821) Database parsing  
(791) Dependencies parsing  
(785) Dependencies parsing  
(773) Forms Parsing  
(772) Database parsing  
(750) Dependencies parsing  
(749) Dependencies parsing  
(748) Forms Parsing  
(747) Database parsing

**Información general**

Estado: Successfully finished  
Tipo: Forms Parsing  
Fecha Inicio: 03/10/2012 16:30:23  
Fecha Fin: 03/10/2012 16:40:56  
Tiempo: ,1 hours or 10,5 minutes  
N. errores: 0  
N. warnings: 6828

**Table:**

Estado	Tipo	Progreso	Tiempo	N. Objetos	N. Objetos Procesados	Fecha inicio	Fecha finalización	N. Warnings	N. Errores
Successfully finished	Builtin forms functions	100%	0 hours or ,2	30	30	03/10/2012 16:30:39	03/10/2012 16:30:52	0	0
Successfully finished	Program units from	100%	0 hours or 0 minutes	1	1	03/10/2012 16:30:39	03/10/2012 16:30:40	0	0
Successfully finished	Program units	100%	0 hours or 1,3	14	14	03/10/2012 16:30:38	03/10/2012 16:31:57	0	0
Successfully finished	Canvases	100%	0 hours or 1,4	14	14	03/10/2012 16:30:38	03/10/2012 16:32:06	0	0
Successfully finished	Blocks	100%	0 hours or 2 minutes	14	14	03/10/2012 16:30:38	03/10/2012 16:34:10	0	0
Successfully finished	Block sources	100%	0 hours or 0 minutes	14	14	03/10/2012 16:30:39	03/10/2012 16:34:15	0	0
Successfully finished	Items	100%	0 hours or ,9	14	14	03/10/2012 16:30:39	03/10/2012 16:35:09	0	0
Successfully finished	Form triggers	100%	0 hours or 2,3	14	14	03/10/2012 16:30:38	03/10/2012 16:37:28	0	0
Successfully finished	Record groups	100%	0 hours or ,5	14	14	03/10/2012 16:30:38	03/10/2012 16:31:12	0	0
Successfully finished	List of values	100%	0 hours or 2,4	14	14	03/10/2012 16:30:38	03/10/2012 16:33:36	0	0
Successfully finished	LOV column mappings	100%	0 hours or ,4	14	14	03/10/2012 16:30:39	03/10/2012 16:35:36	0	0
Successfully finished	Windows	100%	0 hours or ,2	14	14	03/10/2012 16:30:38	03/10/2012 16:30:52	0	0

powered by itC

information about the current process

FIGURE 4.27. Reverse Engineering Processes page.



- **Business Rules Analyzer:** performs the extraction of concepts, fact types, and business rules of the target system. This program is the implementation of the BRE approach presented in Chapter 3.
- **Forms Comparator:** compares two forms (generally two versions of the same form) using the folder-files hierarchies generated by the Forms Parser, and displays the differences between them regarding data blocks and items. Specifically, this program displays if the items were added, removed or visibly changed (for example, if their visibility was changed to hidden).
- **Form Tables Displayer:** this program displays the tables that a list of forms reference to, including the CRUD operation on them.
- **Lines Counter:** counts the total number of LOC of all executable objects of a target system.
- **Paths Displayer:** displays the dependency tree of a list of executable objects. This program does not require refreshing the materialized views.

## 4.4 Further details about ReTool

ReTool supports the processing of multiple target systems. The extractors, analyzers and visualizers have several options to change from one system to another. In the case of the DB, Forms and Dependencies Analyzers the command-line argument `-dbs` is used for selecting the system, while in ReTool Web, the selector in the login page and the header of every page changes the current system. In addition, the tool supports incremental processing of objects, which means that the tool is able to process the selected type of objects instead of processing all types (all the objects). In the same way, it is possible to process and update specific objects of the target system. For example, the analyzers allow to process *Table X*, *Y* and *Z* of a determined target system, through the command-line arguments of the programs.

All the extractors and analyzers are multi-threading, which reduces the processing time of the systems. For an information system as SIFI, the average processing time of the database is about 30 minutes, for the forms is 3 hours, and for the dependencies tree 2.5 hours. In total, the average processing time for SIFI is 6 hours. The time for refreshing the materialized views is about 1-2 hours, depending on the amount of data in the repository and the server load. Apart from this, the analyzers stores data about the processing, including the state, the type of processing, the date and time when it started, the ending date-time, the spent time and the number of errors/warnings. In this way, it is possible to know, for every type of object, the current number of objects processed versus the total number of objects to process, the progress of the processing, its state and the number of errors/warnings. By the way, ReTool is able to process the objects of the target system regardless an error in specific objects; in other words, the tool is fault-tolerant.

Finally, ReTool is designed and implemented in C/C++, Java, several web technologies such as HTML, XML, XSP and JavaScript and with ITC specific technologies. The ReTool's repository is designed for the Oracle 10g RDBMS, and ReTool Web is supported for Apache Tomcat 7.0.22 or later. The main operating system supported for ReTool is Windows XP or later, but it can also work for Linux and other operating systems that

run the Oracle JVM. The unique running restriction is that the Forms Parser only works in a Windows machine with Oracle Forms 6i installed.

## 4.5 Limitations and future enhancements

The ReTool's features and benefits has been described previously. However, ReTool have some limitations that will be addressed in the future. The following are some limitations and enhancements to be done to analyzers and extractors:

- Currently, ReTool detects wrong procedure/function calls when parsing the source code. For example, the usage of cursors with parameters are taken as function calls. Although, the false positives are validated and not stored in the repository, the performance could be improved and the amount of process warnings could be reduced.
- The target system processing time is high, mainly when refreshing the materialized views. We need to consider other alternatives for these views.
- Oracle Reports<sup>5</sup> is the complement technology of Oracle Forms. SIFI has many reports implemented on this technology but ReTool does not support it. In the same way, object libraries are not processed by the tool. We will move forward to perform reverse engineering on these components.
- In ReTool, the target systems need to be updated periodically in order to keep the repository with the system last changes. Currently, this updating process is manual: a user needs to check in the Version Control System (VCS) of the target system to know the changes performed on the objects and used them as an input to ReTool. We will develop an automatic way of doing this, through the integration between ReTool and VCS.
- The ReTool's command-line tools are not easy to use compared to graphical applications such as ReTool Web. In some cases, when there are many changes in the target system and it should be updated in ReTool, the usage of these tools could be very difficult. Then, we will implement a GUI for the extractors and analyzers.

ReTool Web also needs to be enhanced in some aspects:

- The user may want to customize the visualization of the system; for example, change the layout of pages or the size of boxes or the objects color. User customization is a feature for future work.
- Currently, the tool supports a single language: Spanish. We will move forward to implement Internationalization.
- The tool does not allow the edition of Form and DB objects, including their source code and properties. We will implement this feature in the tool.

---

<sup>5</sup>[http://en.wikipedia.org/wiki/Oracle\\_Reports](http://en.wikipedia.org/wiki/Oracle_Reports)

Finally, ReTool should offer, in the future, integration or interoperability with other systems through exchange formats [32] or any other mechanism, an easy installation, dynamic analysis to the call sequences feature, as complement to the static analysis, and a metrics module.

## 4.6 Summary

ReTool is a reverse engineering tool for supporting the maintenance and understanding of Oracle Forms information systems. This Chapter described the tool in detail, including its architecture, components, features, technical design aspects, limitations and future improvements. The next Chapter discusses the results of the qualitative evaluation performed with a group of ITC employees.

---

## Evaluation and Discussion

---

This chapter presents the evaluation of ReTool and the structural Business Rules Extraction technique.

### 5.1 Evaluation of ReTool

We performed a qualitative evaluation of ReTool, through a survey, since we were interested in how the tool has been useful for the people in the company (ITC).

#### 5.1.1 Purpose of the survey

The survey was conducted with the goal of getting feedback from the people that have used the tool, taking into account some aspects such as:

- The usage of several tool features.
- The everyday tasks that the tool has supported.
- The level of gain in time, effort and precision in the understanding and maintenance tasks on SIFI.
- The users' perception about the visual and information quality.

This feedback allowed us to establish how the tool has performed in the maintenance and understanding of SIFI, and also how the quality of the tool is regarding the usage from the users. Also, we were able to detect some improvement aspects that constitute the basis for additional future work.

#### 5.1.2 Subjects

The target people were developers and maintainers of SIFI and SGF<sup>1</sup>. Forty-two people, including managers, functional and technical people constituted this group. The people

---

<sup>1</sup>SGF (Sistema de Gestión Financiera in Spanish) is another Oracle Forms information system owned by ITC. The survey also included this system.

ranged from junior to senior employees (developers and functional people) and from new to old employees in the company. However, we selected a subset of all the potential respondents of the survey, and chose the people that use the tool the most. In total there were 29 respondents, representing the 69% of the target group.

### 5.1.3 Survey description

The survey consisted in 11 questions that can be divided into five categories (the complete questionnaire can be found in the Appendix):

1. Questions to categorize the respondents (2 questions).
2. Questions intended to know the tool usage and the tasks that are supported by the tool (2 questions).
3. Questions that evaluate the level of improvement in time, effort and precision in the user tasks (3 questions).
4. Questions that evaluate some quality attributes of the tool (1 question).
5. Questions intended to know some tool improvements that users considered important to make in the future (3 questions).

### 5.1.4 Results

The survey was answered by 29 people, 79,3% (23 people) of them corresponds to the Software Factory Group, the people that perform adaptive and perfective maintenance on SIFI; 17,2% (5 people) belong to the Support Center Group, the people that perform corrective maintenance; and 3,4% (1 person) to the Functional Group, which is the group that know the business domain of SIFI/SGF. The distribution of the respondents is shown in Figure 5.1.

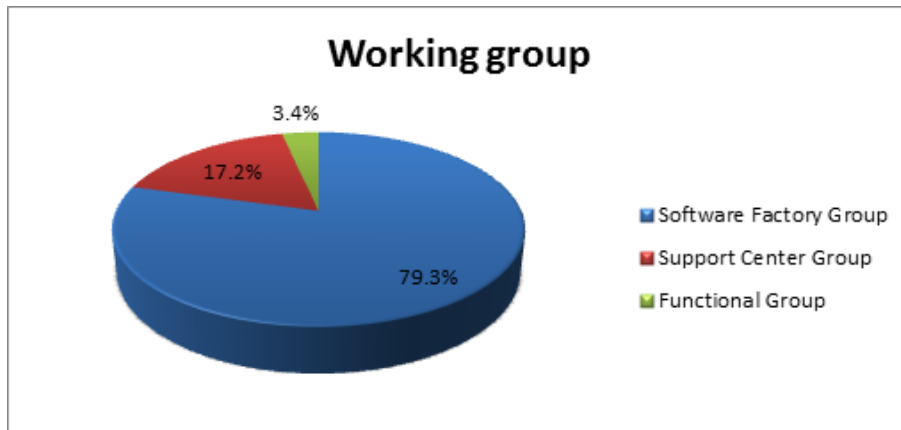


FIGURE 5.1. Distribution of the survey respondents by working group.

We also categorize the respondents by their working time in the company (Figure 5.2). It was found that 44,8% of the people has been working at ITC for more than 2 years, 34,5% have been working between 6 and 12 months, 13,8% between 1 and 2 years, and

the 6,9%, the new people in the company, have been working from less than 6 months. People with more working time in the company are senior developers, while the rest are junior (basically, the novices) or between junior and senior developers.

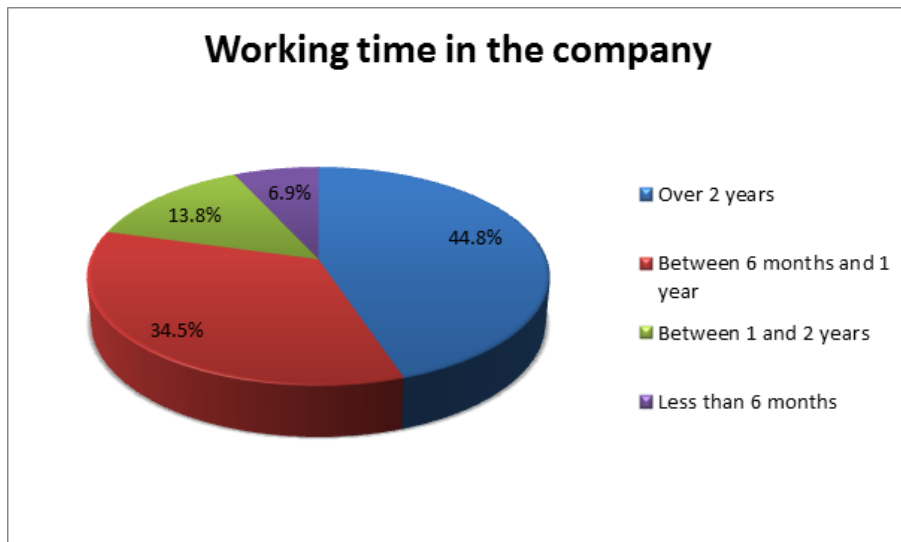


FIGURE 5.2. Distribution of the survey respondents by working time.

Regarding the usage of the tool general features (Figure 5.3), we found that the feature most used is the DB/Forms object searching with 96,6% of usage; in other words, 96,6% of the respondents use this feature. The browsing of object dependencies is used in a 93,1%, the visualization of the objects structure in a 65,5%, the source code visualization in a 41,4%, the extraction of data from a database in a 20,7%, and finally, the information export to Excel files is the least used, with a usage of 3,4%.

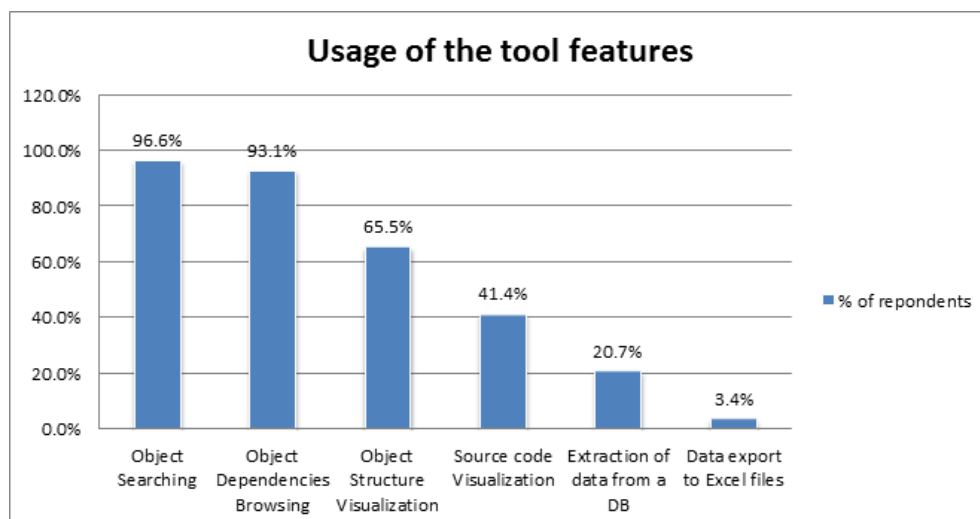


FIGURE 5.3. Level of feature usage of the tool.

Figure 5.4 shows the tasks in which the tool has been used and the degree of usage of the tool in each task. The tool has been mostly used in the measurement of changes impact with 86,2% of the respondents who use the tool for supporting this task. The technical understanding of the system (SIFI/SGF) is next with 62,1% of support. Bug detection is

supported in a 37,9%, followed by bug fixing and planning of maintenance tasks, both with a 34,5% of support. The detection of useless database/Oracle forms objects is next with the 27,6%. The detection of inconsistencies between the design and the implementation is another task for which ReTool has been useful with 17,2%, and finally, the least supported task is the functional understanding of the system with a 6,9%.

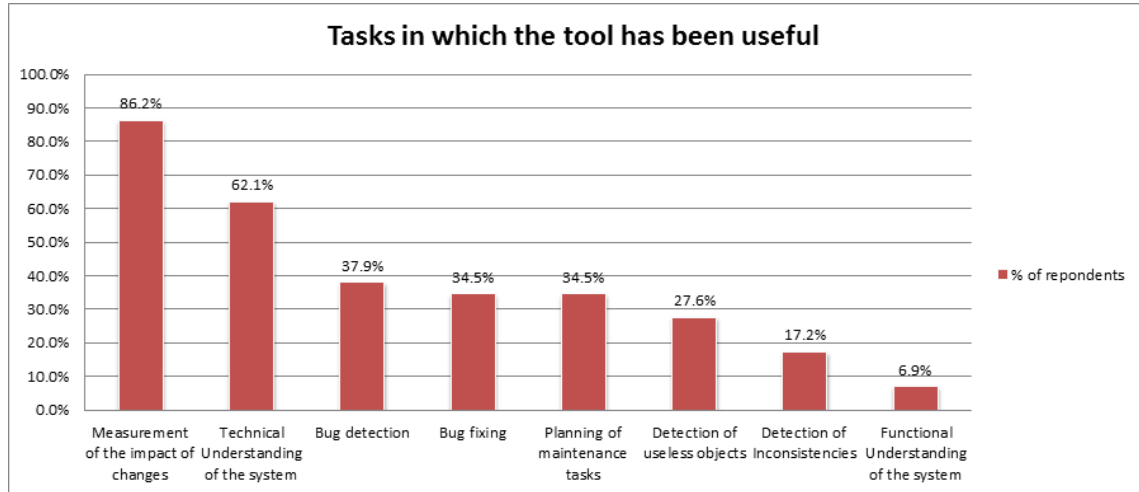


FIGURE 5.4. Level of tool usage on development tasks.

People were also asked about their effectiveness for completing the tasks and artifacts by using the tool, in terms of time (Figure 5.5), effort (Figure 5.6), and precision (Figure 5.7). Only one person could not answer the questions related with this criterion, because he started to use the tool when he started to maintain SIFI, so he did not have comparison criteria.

We found that 14,3% of the respondents said that the degree of tool support in reducing the time to complete tasks is *Very much*, and 71,4% said that this support is *Much*. In contrast, 14,3% said that there was *little* tool support, and nobody said (0%) that the tool was useless at all. Regarding the level of tool support in reducing the effort for completing tasks, 10,7% and 75% of the respondents said that this support has been *Very much* and *Much*, respectively; and 14,4% of them said that the tool was *little* useful in this aspect. Again, no one perceives ReTool as a useless tool. Finally, regarding the precision of tasks and artifacts, 10,7% of the people rated the tool support as *Very much*, 64,3% as *Much*, 25% as *Little*, and 0% as *Nothing*.

We also asked the people to rate some quality attributes of the tool in a 1-4 scale, having 1 as the worst score and 4 as the best one. Figure 5.8 shows the results, having the best scored attribute: “Ways of searching the information” with a score of 3.45, and the worst, but well, scored attributes: “Completeness about the displayed information” and “Level of detail about the displayed information” with a score of 3. Among them are, in decreasing order, “Speed in the information displaying” with 3.29, “Intuitive information visualization” with 3.28, “Ways of browsing the DB/Forms objects” with 3.24, “Ways of visualize the information” with 3.21, and “Tool usability” with 3.07.

Finally, the survey respondents gave us several comments and suggestions about improvements of the tool regarding the information it provides or could provide, and the way of visualizing such information. We received 39 suggestions that were classified into several categories. However, some suggestions were not considered in this categorization

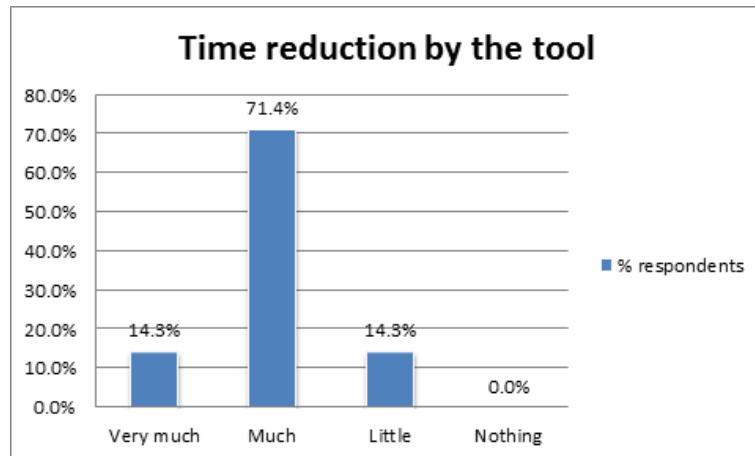


FIGURE 5.5. Degree of tool support in reducing the time to complete tasks.

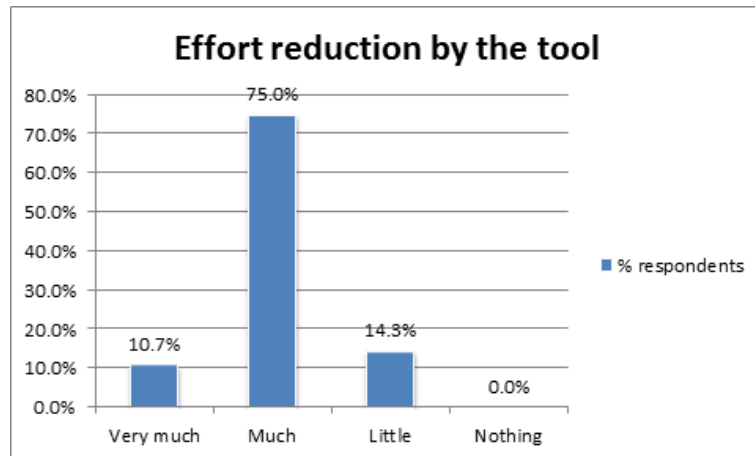


FIGURE 5.6. Degree of tool support in reducing the effort to complete tasks.

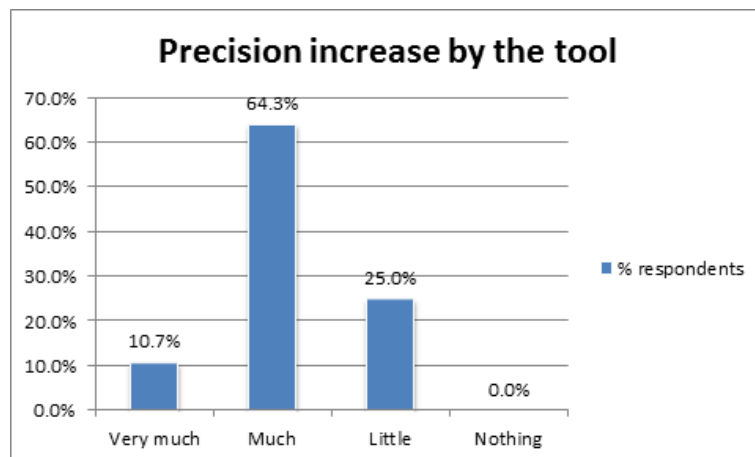


FIGURE 5.7. Degree of tool support in increasing the precision to complete tasks and artifacts.



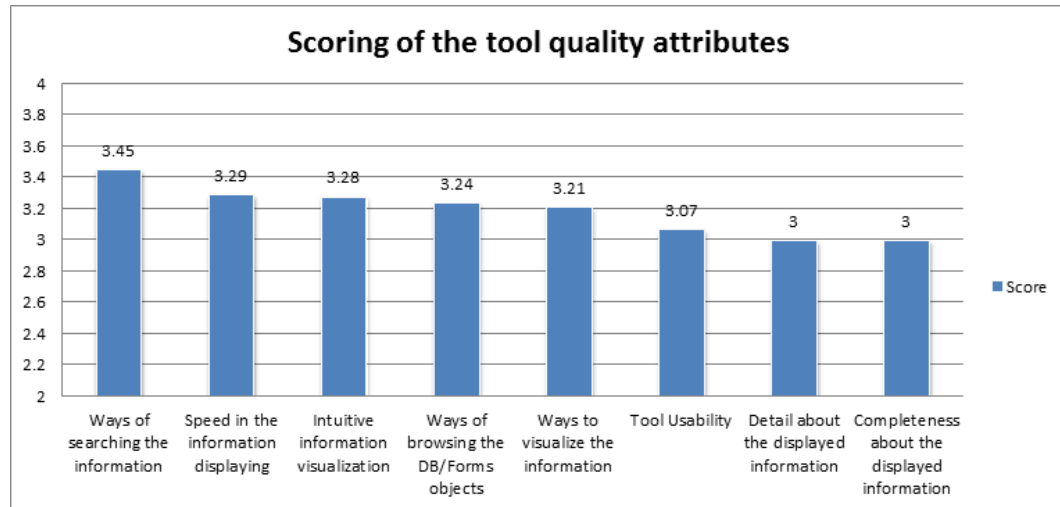


FIGURE 5.8. Rating of some quality attributes of the tool.

(Figure 5.9) because they were beyond the scope of the tool (1 suggestion) so they were *Not accepted*, some of them were not clear so we did Not understand them (3 suggestions), and some of them contained comments about features and attributes that were already *Provided* by the tool (7 suggestions). In this way, we classified 28 suggestions in the following categories (Figure 5.10):

1. Eleven of them (39,3%) were about improvements in the tool usability,
2. Nine (32,1%) about the requirement for additional information that the tool should provide,
3. Three (10,7%) about providing more semantic information,
4. Two (7,1%) about additional filters,
5. and there was one suggestion about code instrumentation, one about improving the speed of searching and one about improving the searching feature (3,6% each).

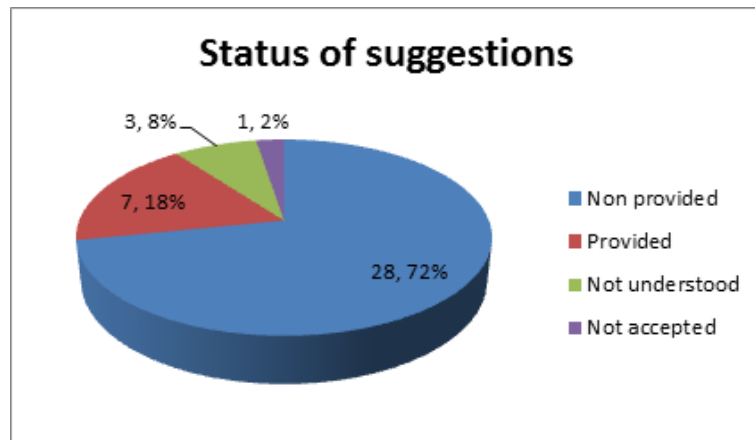


FIGURE 5.9. Status of the received suggestions from the respondents.

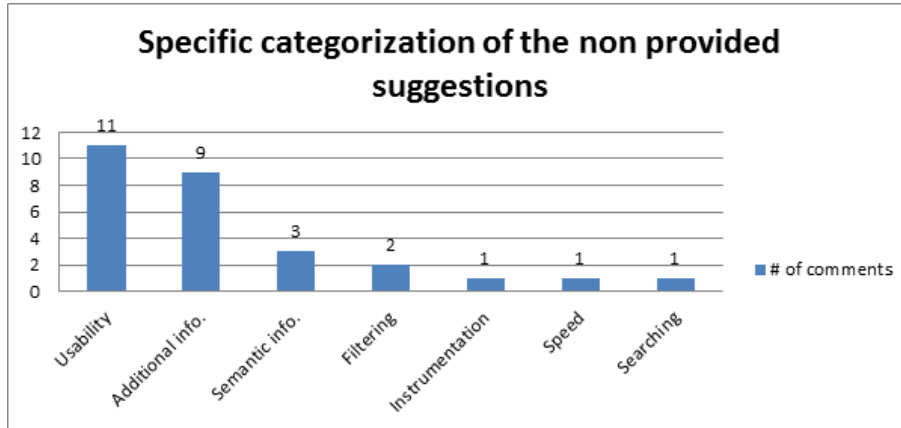


FIGURE 5.10. Specific categorization of the non provided suggestions.

The Usability and Filtering categories correspond to “Visualization suggestions”, Additional and Semantic Info. categories to “Information suggestions”, and the others to “Feature suggestions”. In this way, we received 13 “Information suggestions”, 13 “Visualization suggestions” and 2 “Feature Suggestions” (Figure 5.11).

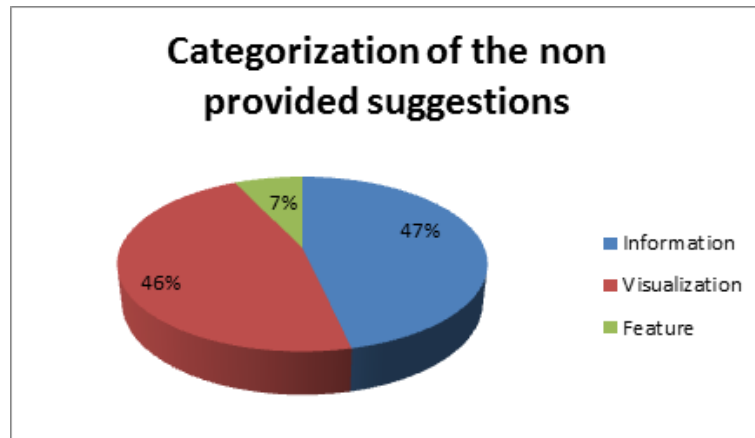


FIGURE 5.11. Categorization of the non provided suggestions.

### 5.1.5 Analysis and Discussion

Most of the tool users are technical, i.e., developers and maintainers. There are few functional people using the tool, and we asked only one functional employee to answer the survey. We think this tool is intended mainly for technical people because all the information that tool displays is about technical aspects of the target systems. However, we found that people want more semantic and functional information, which was manifested in some suggestions. This can also be reflected on the low tool usage for the task “Functional understating of the system” (Figure 5.4). In any case, junior and senior developers use the tool as well as new and old employees<sup>2</sup>.

<sup>2</sup>By the way, all senior developers have a long working time in the company, while the junior ones a short time.

Regarding the usage of the tool's general features (Figure 5.3), the most important features have a high level of usage as expected: "Object Searching", "Dependencies Browsing" and "Structure visualization". However, we expected more usage of "Source code visualization". In the suggestions, we received an improvement request about this feature, and we think the main reason for this low usage is that the code cannot be edited in the tool, and the visualization is limited to highlighting and line numbering. On the other hand, we expected low usage of Extraction of data from a DB because it is a very specific feature and is not necessary for all the people in the company. The usage of Data export to Excel files is also low because it is a new feature, so most of the people have not used it yet. In general, there is a high level of usage of the tool.

The tasks that the tool supported the most were "Measurement of the changes impact" and "Technical understanding of the system" (Figure 5.4). This was expected because of the large size and the high coupling of SIFI and SGF, and the fact that there is no documentation about the systems architecture and domain, so these tasks are very common in the company. As stated before, the "Functional Understanding of the system" is the less supported but required task; the rest of the tasks have been supported by the tool in some cases.

Regarding the effectiveness of tasks, people perceive ReTool as a very useful tool for supporting their tasks (Figures 5.5, 5.6 and 5.7): 85,7% assigned a positive usefulness in time and effort reduction versus 14,3% of negative usefulness, and 75% of positive usefulness for increased precision versus 25% of negative precision. One of the reasons for the negative perception of usefulness is the usage level of the tool by some people. The tool usage varies because some people have roles in the development group that only require the tool in specific scenarios, and in some cases, there are people that have started using the tool.

In general, the perceived tool quality attributes by the users is high. The tool has several ways of searching the information, the speed of the information displaying is high, the information visualization is intuitive, it has several ways of visualizing and browsing the system objects, the tool is easy to use, it presents enough detail about the information, and the completeness about the information is high. However, the users suggested several specific improvements about some attributes and features:

- The tool should allow manual or automatic semantic clustering of objects, and extraction of high level processes.
- In several cases, the granularity should be more specific. ReTool granularity is at procedure level.
- The tool should calculate quality metrics of the target systems.
- The tool should include processing and displaying of additional DB and Forms objects, such as sequences or pop-up menus.
- The information visualization should be more integrated and synchronized, and also more specific through filters.
- The tool should be more intuitive regarding the searching and filtering feature.

Other suggestions were more specific regarding the target system. For example, one user said: *"The tool should display the version number of the objects"*. In SIFI/SGF, each

object has a number that corresponds to the version, but this specific implementation decision on these systems. ReTool could work for other Oracle Forms systems different from SIFI and SGF.

In conclusion, ReTool seems to be very useful in the understanding and maintenance of Oracle Forms applications. It improves the productivity of developers to complete their tasks through several functionalities such as object searching, visualization and browsing of object dependencies and structure, visualization of source code, data exporting from a database to INSERT scripts, and export of the displayed information to Excel files. The tool has been used in the company for more than one year and it has been successful. On the other hand, the tool needs to be improved in order to cover some aspects and requirements that the developers and the functional people request.

## 5.2 Evaluation of the BRE technique

ReTool implemented the algorithm and DB-BR mappings and heuristics, presented in Chapter 3, and it is able to extract:

- Concepts from tables and columns, using the comments in the DB.
- Verbs and binary property fact types from the table/column comments and concepts, and from the tables hierarchy (foreign keys).
- Categorization and unary fact types from check list constraints.
- Structural BR from the extracted fact types, using the non null constraints.

The input of the tool is a list of DB tables and the output is a set of extracted and stored concepts, verbs, fact types, sentences and business rules. The tool processed the DB components of the input tables and the tables related to them through the foreign keys (parent and children tables) and created the output in about 6 minutes of execution time. Table 5.1 presents some examples of the resulting components. All the extracted BRs and components can be downloaded from <http://www.itc.com.co/WCRE12/3>.

Table 5.2 summarizes the results of the BRE tool in the PP module. The 25 core tables of the module were the input to the tool, resulting in 155 processed tables and 3,447 columns, for a total of 3,602 processed objects. The number of extracted concepts from tables and columns was 2,508, of which 2,142 correspond to general concepts, i.e., concepts of tables and columns, and 366 correspond to categorical concepts, i.e., concepts extracted from columns involved in check list constraints, which values represent classes of concepts. In addition, the tool was able to extract 413 verb phrases and a total of 4,005 fact types using the extracted concepts, verbs, and table constraints. 488 unary fact types were extracted (from check list constraints, whose values represent states of concepts), and also 366 categorization fact types. The number of extracted binary fact types is 3,517; 114 of those were extracted from foreign keys having verbs in the column comments, and 177 are with no verbs in comments. At the end, 870 sentences and structural BRs were generated using non null constraints and the extracted fact model. 22% of fact types were used to create the structural BRs.

<sup>3</sup>The dataset is available only in Spanish.

TABLE 5.1. Examples of the extracted BR components from the PP module.

BR component	Examples
<u>Concept</u>	<i>Trust Payment Movement, Operation Check</i>
<u>Unary Fact Type</u>	<i>Treasury Movement Type is tax-exempt</i>
<u>Binary Fact Type</u>	<i>Expenditure per Contribution generates Treasury Movement</i>
<u>Property Fact Type</u>	<i>Credit Note has Specific Voucher Type</i>
<u>Categorization Fact Type</u>	<i>Fixed Investment Depreciation is a category of Investment Depreciation</i>
<u>Structural Business Rule</u>	<i>Every Deposit always has a Transaction Value</i>

### 5.2.1 Evaluation

Once we refined the algorithm and obtained these results, we conducted a study with four of the ITC employees to evaluate the precision of the results, i.e., we wanted to identify the false positives. At this stage, we cannot assess the recall of the tool, as we do not know exactly how many BRs are encoded in the system and in the database, hence we cannot estimate how many we are missing. Since this work is in an industrial context, usability is critical. Hence, we wanted to make sure that the precision is not too low and the future users will not have to investigate too many false positives, as that would prove to be a deterrent in using this tool. This is why we introduced some of the conservative heuristics described in Chapter 3. After discussing with the maintainers and managers of the system, we decided that the lowest acceptable precision would be around 20-25%. In other words, the people interested in the knowledge acquisition were not willing to look at more than 3-4 wrong rules in order to get a good one (in average).

We selected a subset of the rules extracted by the tool, and asked four employees of the company to analyze them. The four subjects know well the functionality of the software system, the module where the business rules were extracted from, and have extensive experience in the fiduciary business. The objects of the study were 300 rules randomly selected from the set of 870 rules extracted by the tool (34%).

Specifically, each expert told us if the rules extracted actually embody business knowledge or just system implementation details, or if they do not make sense and why. Since all the employees have high level of knowledge of the system and the fiduciary business, each rule was evaluated by only one human expert. Given the limited time resources, we decided to ensure higher coverage of the results, rather than ensure redundancy to avoid evaluation mistakes. Thus, each subject judged 75 rules by answering some questions about them. Given a business rule  $R$ , the following questions were asked to each subject:

- Does the rule  $R$  make sense?

TABLE 5.2. Statistics of the extracted BRs and components from the PP module.

Statistic	Value
# of processed objects (tables/columns)	3,602
# of processed tables	155
# of processed columns	3,447
# of concepts	2,508
# of general concepts (tables/columns)	2,142
# of categorical concepts	366
# of verbs	413
# of fact types (FT)	4,005
# of unary FT	488
# of binary FT	3,517
# of property FT	3,151
# of categorization FT	366
# of FT from FK (verbs)	114
# of FT from FK (no verbs)	177
# of sentences	870
# structural BR	870
% of BR from # of FT	22

- If yes, is  $R$  a rule of the fiduciary business (domain rule) or a rule of the software system (implementation rule)?
- If not, please specify what you do not understand about the business rule  $R$ .

Of the 300 extracted rules, 195 (65%) of the rules in the sample were deemed correct by the subjects, while 105 (35%) were considered incorrect (see Table 5.4a). Within the correct rules, 87 (29%) were judged as business rules, and 108 (36%) as implementation rules (see Table 5.4b). Overall, the subject were satisfied with the precision of the tool. In average, one in three rules was a good business rule and one more was a correct implementation rule, which the subjects also deemed useful (although our focus was strictly on the BRs). We focused our attention on the false positives (i.e., the 105 rules deemed incorrect).

TABLE 5.3. Results of the BR evaluation.

<b>Total Correct Rules</b>	195 (65%)	<b>Correct Business Rules</b>	87 (29%)
<b>Total Incorrect Rules</b>	105 (35%)	<b>Correct Implementation Rules</b>	108 (36%)

(A) Number of correct and incorrect rules.

(B) Number of correct business and implementation rules.

As mentioned, we asked the subject to point out (where possible) or speculate what is wrong with the rules. Table 5.4 summarizes the obtained information. 57 of these rules were incorrect because their concepts missed some terms essential for their understanding, or had unnecessary terms that make them difficult to understand. We also found that 7 rules made no sense because the same concept was repeated twice in the rules. In addition,

44 rules contained unclear concepts. We found that this lack of clarity comes from the comments of tables and columns, from where these ill-defined concepts were extracted by the tool. Among these incorrect rules, we found that 17 rules contain operative concepts that should be used for operative rules instead of structural business rules. In these cases, although the concepts were understood, the rules were incorrect because the keywords that restricted the fact types are wrong. 2 of the incorrect rules correspond to real business rules (if fixed) and 13 to implementation rules (if fixed). An incorrect rule may be classified in several categories of Table 5.4.

TABLE 5.4. Quantitative information about the misunderstood/incorrect rules.

Rules	Num.
Miss/add Terms	57 (54%)
No-sense Rules	7 (7%)
Wrong Comments	44 (42%)
Operative Concepts	17 (16%)
Implementation Rules	13 (12%)
Business Rules	2 (2%)

In summary, we detected three elements that negatively affect the precision of the BRE approach:

1. Poor quality in some table/column comments for the extraction of verbs, concepts, categorization, and unary fact types. In the case of categorization and unary fact types, the existence of table/column comments is mandatory. If comments do not exist in the DB, the mapping of all BR would be manual or would be extracted from other sources of information. We found comments with no natural language, such as, “—” or “\*\*\*\*”; comments that specify examples of the concepts instead of the concepts; comments with incomplete descriptions that miss important terms describing the concepts; comments with unnecessary and confusing information; comments with erroneous descriptions of the tables and columns; and comments with spelling errors. Our tool could be also used to point out places where the comments need to be updated.

2. The precision of the POS tagger, which leads to missing or wrong detected nouns and verbs in comments. In the same way, this leads to incomplete extraction of concepts, verbs and fact types. We plan to try out other POS taggers in the future. In some cases, the POS tagger was misled due to misspellings. We plan to run a spell checker to clean the comments in the future. A more complex natural language processing of the comments may be also needed.

3. Decayed architecture of the PP module as a result of the long evolution of the system. This is reflected in the number of nullable columns and the presence of parametric and characteristic columns<sup>4</sup> in the same table. In some cases, this leads to the extraction of false structural BRs. In a few cases the tool creates non sense rules where a concept is associated with itself, such as the rule: “*Every Deposit always has a Deposit*”. This happens with columns having a non null constraint and a primary key constraint. We will modify our algorithm to exclude these cases. Additionally, columns having a non null constraint and a check list constraint that verify ‘S’ (Yes) and ‘N’ (No) values were not excluded. As explained in section 3.2.1, they are used for guiding the operations

<sup>4</sup>Characteristic columns are those representing basic information of the table’s entity.

and processes in the system, instead of structuring and organizing knowledge about the business.

### 5.3 Summary

A survey was conducted with a group of ITC employees to know how the ReTool has been useful for them. The results show that tool is very useful in the understanding and maintenance of SIFI and SGF, as it improves the productivity of developers to complete their tasks and the maintenance process of the systems now is easier. On the other, the proposed BRE approach was assessed. We conducted a study with four of the ITC employees to evaluate the precision of the results, i.e., the correct BRs, extracted by the tool. We found that 29% of recovered rules are correct structural business rules, 36% correspond to implementation rules, and 35% are incomplete or incorrect rules. The results show that the recovery technique is practical, while there is room for improvement, and it will be used as basis for the recovery of additional knowledge.



---

## Conclusions

---

This thesis presents an Oracle Forms reverse engineering tool, called ReTool. The tool aims at supporting the understanding and the maintenance of legacy information systems, by providing automatic processing, searching and visualization of the system object dependencies and structure. Moreover, the tool provides several features and attributes that make it a versatile and useful tool. The evaluation shows the potential of the tool regarding the successful use case in the industry. The tool was applied to SIFI, a large and complex financial legacy system, in which the tool supported several maintenance and understanding problems of the system, that developers address every day.

Furthermore, we presented an approach for automatically extracting structural business rules from legacy databases and its application on a specific legacy system. Specifically, we performed a revision of the BR concepts based on the SBVR standard, proposed an approach that considers DB-BR component mappings to extract structural BRs from databases, applied the approach in an industrial legacy information system, and performed a preliminary assessment of the extracted components and rules. The results showed that 29% of the extracted rules are correct business rules and 36% are correct implementation rules. The tool was deemed useful and usable by the evaluators. The evaluation study revealed some of the causes for the false positives, which we plan to address in future work (at least in part). While some aspects of our BRE algorithm are generic, most of the heuristics it uses are specific to the systems we worked on. The generalization of our solution is beyond the scope of this work, yet we believe that our experience can be adapted to other existing systems.

### 6.1 How does ReTool support the maintenance and understanding of SIFI?

ReTool implements several variants of reverse engineering standard techniques (see Section 2.2.1). ReTool creates structure trees and calls/dependencies graphs that are modeled by the ReTool's repository metamodel. The visualization of this information is achieved through a web application, in which there are boxes that contain different information that result of specific queries, useful for the developers in their daily tasks. The boxes layout in several pages organize and present the information to the user, in an on-demand and scalable way. The object searching feature allows the user to perform quick searches and also advanced searching. This feature also allows the user to locate objects of interest, and obtain related information with specific objects or concepts. The page navigation and

browsing of object dependencies and structure, allows the user to inquire about dependencies and change effects on objects and also to understand how the target system works technically. Additionally, the distributed character of the visualizer encourages the users to test and use it, as it needs to be installed only in one machine.

Regarding the specific problems of SIFI, ReTool provides several ways to support its maintenance and understanding from developers perspective. All these advantages are evidenced by the ReTool evaluation in Chapter 5:

- ReTool processes the dependencies of objects automatically in SIFI. Before ReTool, this job was manual and tedious, because the users had limited tools for this: the text searching option of development environments and tools such as the Oracle Form Builder<sup>1</sup>.
- ReTool allows the search of Forms and DB objects faster than other tools (for example, the Oracle Form Builder or PL/SQL Developer<sup>2</sup>) and in an integrated way. This means that users only need one tool for searching objects. Before ReTool, users needed at least two tools for searching: one for Oracle Forms and one for the Oracle Database.
- ReTool presents the information about SIFI in an organized, condensed and graphical way. This allows the users to have different focus points, regarding the information they need, to get synthesized information, to perform a structured maintenance/understanding process and to obtain a visual representation of the software system. The information visualization of SIFI is very important as it addresses the invisibility and complexity of the system [21][33].
- The BRE approach, implemented in ReTool, is a starting point for re-documenting the business rules that implement SIFI.

The cost of all the ReTool advantages is the installation/configuration of its components and the processing time of the target system.

## 6.2 Future Work

As future work we will continue developing ReTool, regarding all the future enhancements that were presented in Section 4.5.

Additionally, we plan to implement and refine all the proposed DB-BR mappings. The refinement will include:

- The analysis of temporary tables and check constraints with any type of conditions.
- The detection of roles in fact types [44], verb phrases from tables that only relate other tables (many-to-many relationship), and concept synonyms.
- The extension of the BR format/scheme, in order to represent several type of rules in other forms, e.g., decision tables as a way for representing parallel business rules [44].

---

<sup>1</sup>See <http://www.oracle.com/technetwork/documentation/6i-forms-084462.html> and [17] to get information about this tool

<sup>2</sup><http://www.allroundautomations.com/plsqldev.html>

We also plan to perform a formal qualitative/quantitative evaluation of the approach in terms of precision and recall, and finally, we will move forward to the extraction of operative business rules.

### 6.3 Recommendations about the maintenance of SIFI

This thesis concludes by proposing some recommendations about how the maintenance process on SIFI should be addressed by the company, based on the experience of the author and the application of reverse engineering on it using ReTool:

- We encourage the company to perform perfective maintenance on SIFI, in an automatic way if possible. Of course, this is not an easy task, but, at least, it should be applied to those Oracle Forms/Database objects that are critical due to their coupling.
- There should be a detailed strategy for this, including specific procedures and aspects such as software metrics and manual or automatic testing, that assures an effective refactoring. Reverse engineering and software maintenance tools such as Retool, are designed for making this task easier.
- In this sense, the adoption of useful maintenance tools is essential. Tools for concept location, impact analysis, automatic testing, metrics, and so on, need to be included in the SIFI maintenance process. We know that there are few tools for Oracle Forms applications, but it is possible to build them. ReTool is one example and one important step for trying to improve the process on this technology.
- We also suggest the company to define and adopt development and maintenance standards about SIFI in its specific context. Detailed procedures, methodologies, roles, best practices and tools should be clearly defined and enacted.

## APPENDIX

---

---

### ReTool Evaluation survey

---

---

ReTool is an application that supports the process of understanding and maintenance of SIFI/SGF. This survey aims to evaluate the usefulness of the tool, from the use and perception of people working in ITC. The results will improve the tool, in order to support even more the understanding and maintenance activities in SIFI/SGF. Please answer the following questions.

1. What is your working group in ITC?
  - ☐ Functional Group (Investments Funds, Financial Investments, etc.)
  - ☐ Technical Group of Investments Funds
  - ☐ Technical Group of Financial Investments
  - ☐ Financial Technical Group
  - ☐ Technical Group of Trusts
  - ☐ Transversal Technical Group
  - ☐ Web Technical Group
  - ☐ Support Center Group
  - ☐ Other: \_\_\_\_\_
2. How long ago do you work in ITC?
  - ☐ Less than 6 months
  - ☐ Between 6 months and 1 year
  - ☐ Between 1 and 2 years
  - ☐ Over 2 years
3. What ReTool features have you used?
  - ☐ Forms and DB Object Searching
  - ☐ Object Dependencies Browsing
  - ☐ Data extraction from the target database, through the generation of INSERT statements

- 
- ☐ Object (forms, tables, packages, etc.) Structure Visualization
  - ☐ Source code Visualization
  - ☐ Others:
    - \_\_\_\_\_
    - \_\_\_\_\_
4. In your daily work in ITC, the tool has helped you to (check all that apply):
- ☐ Understand how SIFI/SGF works technically
  - ☐ Understand how SIFI/SGF works functionally
  - ☐ Detect SIFI/SGF defects
  - ☐ Fix SIFI/SGF defects
  - ☐ Detect implemented vs. designed inconsistencies
  - ☐ Detect useless or unused objects, or objects that should be removed
  - ☐ Measure the impact of changes of SIFI/SGF
  - ☐ Plan maintenance activities in SIFI/SGF
  - ☐ Others:
    - \_\_\_\_\_
    - \_\_\_\_\_
5. What is the degree of tool support in reducing the time to complete your tasks?  
Choose one of the following options.
- ☐ Very much
  - ☐ Much
  - ☐ Little
  - ☐ Nothing
6. What is the degree of tool support in reducing the effort to complete your tasks?  
Choose one of the following options.
- ☐ Very much
  - ☐ Much
  - ☐ Little
  - ☐ Nothing
7. What is the degree of tool support in enhancing the precision to complete your tasks and artifacts? Choose one of the following options.
- ☐ Very much
  - ☐ Much
  - ☐ Little
  - ☐ Nothing
8. Rate the following attributes of the tool from 1 to 4, being 4 the highest score and 1 the lowest

- 
- Intuitive information visualization \_\_\_\_\_
  - Tool Usability \_\_\_\_\_
  - Ways of visualize the information \_\_\_\_\_
  - Ways of searching the information \_\_\_\_\_
  - Ways of browsing the DB/Forms objects \_\_\_\_\_
  - Level of detail about the displayed information \_\_\_\_\_
  - Completeness about the displayed information \_\_\_\_\_
  - Speed in the information displaying \_\_\_\_\_
9. What improvements do you think should be done to the tool, regarding the displayed information?
10. What improvements do you think should be done to the tool, regarding the way the information is displayed?
11. Write down the features, modules or pages of the tool in which you require personalized training.

---

---

## Bibliography

---

---

- [1] S. Ali, B. Soh, and J. Lai, *Rule extraction methodology by using XML for business rules documentation*, Industrial Informatics, 2005. INDIN '05. 2005 3rd IEEE International Conference on, August 2005, pp. 357 – 361.
- [2] P. Álvarez and J. A. Bañares, *Sistemas de Información Distribuidos, conceptos y estándares de arquitecturas orientadas a servicios Web.*, 2006.
- [3] L. Andrade, J. Gouveia, M. Antunes, M. El-Ramly, and G. Koutsoukos, *Forms2Net - Migrating Oracle Forms to microsoft .NET*, Generative and Transformational Techniques in Software Engineering (R. Lämmel, J. Saraiva, and J. Visser, eds.), Lecture Notes in Computer Science, vol. 4143, Springer Berlin Heidelberg, 2006, pp. 261–277.
- [4] N. Asif, *Software reverse engineering process: Factors, elements and features*, International Journal of Library and Information Science **2** (2010), no. 7, 124–136.
- [5] I.S. Bajwa, B. Bordbar, and M. Lee, *SBVR vs OCL: A comparative analysis of standards*, Multitopic Conference (INMIC), 2011 IEEE 14th International, IEEE, December 2011, pp. 261 –266.
- [6] I.S. Bajwa, M. Lee, and Bordbar B., *SBVR Business Rules Generation from Natural Language Specification*, AAAI 2011 Spring Symposium, March 2011, pp. 2–8.
- [7] I. Baxter and S. Hendryx, *A Standards-Based Approach to Extracting Business Rules*, OMG's Architecture Driven Modernization Workshop, 2005.
- [8] I.D. Baxter and M. Mehlich, *Reverse engineering is reverse forward engineering*, Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on, IEEE, 1997, pp. 104–113.
- [9] K.H. Bennett and V.T. Rajlich, *Software maintenance and evolution: a roadmap*, ICSE '00: Proceedings of the Conference on The Future of Software Engineering, ACM, 2000, pp. 73–87.
- [10] I. Burnstein and K. Roberson, *Automated chunking to support program comprehension*, Program Comprehension, 1997. IWPC '97. Proceedings., Fifth International Workshop on, IEEE, 1997, pp. 40–49.
- [11] I. Burnstein, R. Saner, and Y. Limpiyakorn, *Using an artificial intelligence approach to build an automated program understanding/fault localization tool*, Tools with Artificial Intelligence, 1999. Proceedings. 11th IEEE International Conference on, IEEE, 1999, pp. 69 –76.

- 
- [12] G. Canfora and A. Cimitile, *Software Maintenance*, vol. 2, ch. 2, pp. 15–20, World Scientific Pub. Co, 2002.
  - [13] G. Canfora and M. Di Penta, *New Frontiers of Reverse Engineering*, FOSE '07: 2007 Future of Software Engineering, IEEE Computer Society, 2007, pp. 326–341.
  - [14] E. Chikofsky and J. Cross II, *Reverse Engineering and Design Recovery: A Taxonomy*, *Software*, IEEE **7** (1990), no. 1, 13–17.
  - [15] B. Cornelissen, *Evaluating Dynamic Analysis Techniques for Program comprehension*, Ph.D. thesis, Delft University of Technology, 2009.
  - [16] B. Cornelissen, L. Moonen, and A. Zaidman, *An Assessment Methodology for Trace Reduction Techniques*, Proceedings of the 24th International Conference on Software Maintenance, IEEE Computer Society, September 2008, pp. 107–116.
  - [17] Oracle Corporation, *Oracle Forms Developer, Form Builder Reference, Volume 1*, Tech. report, A73074-01, 2000.
  - [18] ———, *Using the Oracle Forms Application Programming Interface (API)*, Tech. report, EIT-DE-WP-56, 2000.
  - [19] R. Cerie, F. Baião, and F. Santoro, *Identificacao de regras de negocio utilizando mineracao de processos*, Companion Proceedings of the XIV Brazilian Symposium on Multimedia and the Web, ACM, 2008, pp. 241–246.
  - [20] J.H. Cross, T.D. Hendrix, and S. Maghsoodloo, *The control structure diagram: An overview and initial evaluation*, *Empirical Software Engineering* **3** (1998), no. 2, 131–158.
  - [21] S. Diehl, *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*, Springer, 2007.
  - [22] P. Dugerdil, *Using trace sampling techniques to identify dynamic clusters of classes*, Proceedings of the 2007 conference of the center for advanced studies on Collaborative research, ACM, 2007, pp. 306–314.
  - [23] A.B. Earls, S.M. Embury, and N.H. Turner, *A method for the manual extraction of business rules from legacy source code*, *BT Technology Journal* **20** (2002), 127–145.
  - [24] H. Gall, R. Klösch, and R. Mittermeir, *Abstract Pattern-Driven Reverse Engineering*, Development and Evolution of Software Architectures for Product Families, Second International ESPRIT ARES Workshop, 1995, pp. 334–341.
  - [25] S. Ghandeharizadeh and J. Yap, *Materialized Views and Key-Value Pairs in a cache Augmented SQL System: Similarities and Differences*, Tech. report, USC Database Laboratory, 2002.
  - [26] Object Management Group, *Semantics of Business Vocabulary and Business Rules (SBVR), v1.0*, 2008.
  - [27] J.L. Hainaut, A. Cleve, J. Henrard, and J.M. Hick, *Migration of Legacy Information Systems*, Software Evolution, Springer Berlin Heidelberg, 2008, pp. 105–138.



- 
- [28] A. Hamou-Lhadj, E. Braun, D. Amyot, and T. Lethbridge, *Recovering Behavioral Design Models from Execution Traces*, Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on, IEEE, 2005, pp. 112 – 121.
  - [29] D. Hay and K. Healy, *Defining Business Rules - What Are They Really?*, July 2000.
  - [30] H. Huang, W.T. Tsai, S. Bhattacharya, X.P. Chen, Y. Wang, and J. Sun, *Business rule extraction from legacy code*, Computer Software and Applications Conference, 1996. COMPSAC '96., Proceedings of 20th International, August 1996, pp. 162 –167.
  - [31] A.C. Kalsing, G.S. do Nascimento, C. Iochpe, and L.H. Thom, *An Incremental Process Mining Approach to Extract Knowledge from Legacy Systems*, Enterprise Distributed Object Computing Conference (EDOC), 2010 14th IEEE International, IEEE, October 2010, pp. 79 –88.
  - [32] H. M. Kienle and H. A. Müller, *The Tools Perspective on Software Reverse Engineering: Requirements, construction, and Evaluation*, Advances in Computers (M.V. Zelkowitz, ed.), Advances in Computers, vol. 79, Elsevier, 2010, pp. 189 – 290.
  - [33] R. Koschke, *Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey*, Journal of Software Maintenance and Evolution: Research and Practice **15** (2003), no. 2, 87–109.
  - [34] D. Lo, *Mining specifications in diversified formats from execution traces*, Software Maintenance, 2008. ICSM 2008. IEEE International Conference on, IEE, 2008, pp. 420–423.
  - [35] D. Lo, S.C. Khoo, and C. Liu, *Mining past-time temporal rules from execution traces*, Proceedings of the 2008 international workshop on dynamic analysis, ACM, 2008, pp. 50–56.
  - [36] C. Lu, W. Chu, C. Chang, Y. Chung, X. Liu, and Yang H., *Reverse Engineering*, vol. Vol. 2, ch. 18, pp. 447–466, World Scientific Pub. Co, 2002.
  - [37] S. Mancoridis, B.S. Mitchell, Y. Chen, and E.R. Gansner, *Bunch: a clustering tool for the recovery and maintenance of software system structures*, Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on, IEEE, 1999, pp. 50 –59.
  - [38] B.S. Mitchell and S. Mancoridis, *On the automatic modularization of software systems using the Bunch tool*, Software Engineering, IEEE Transactions on **32** (2006), no. 3, 193 – 208.
  - [39] T. Morgan, *Business Rules and Information Systems: Aligning IT with Business Goals*, Addison-Wesley Professional, 2002.
  - [40] H.A. Müller, S.R. Tilley, and K. Wong, *Understanding software systems using reverse engineering technology perspectives from the Rigi project*, Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering - Volume 1, 1993, pp. 217–226.
  - [41] E. Putrycz and A. Kark, *Recovering Business Rules from Legacy Source code for System Modernization*, Advances in Rule Interchange and Applications (A. Paschke and Y. Biletskiy, eds.), Lecture Notes in Computer Science, vol. 4824, Springer Berlin / Heidelberg, 2007, pp. 107–118.

- 
- [42] ———, *Connecting Legacy code, Business Rules and Documentation*, Rule Representation, Interchange and Reasoning on the Web (N. Bassiliades, G. Governatori, and A. Paschke, eds.), Lecture Notes in Computer Science, vol. 5321, Springer Berlin / Heidelberg, 2008, pp. 17–30.
  - [43] A. Quilici, *A memory-based approach to recognizing programming plans*, Commun. ACM **37** (1994), 84–93.
  - [44] R.G. Ross, *Business Rule Concepts: Getting to the Point of Knowledge*, third ed., Business Rule Solutions Inc, 2009.
  - [45] H. Safyallah and K. Sartipi, *Dynamic Analysis of Software Systems using Execution Pattern Mining*, Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on, 2006, pp. 84–88.
  - [46] S. Shekar, J. Hammer, M. Schmalz, and O. Topsakal, *Knowledge Extraction in the SEEK Project Part II: Extracting Meaning from Legacy Application code through Pattern Matching*, Tech. report, University of Florida, 2003.
  - [47] T. Skramstad and M.K. Khan, *Assessment of reverse engineering tools: A MECCA approach*, Assessment of Quality Software Development Tools, 1992., Proceedings of the Second Symposium on, IEEE, May 1992, pp. 120–126.
  - [48] H.M. Sneed and K. Erdos, *Extracting business rules from source code*, Program Comprehension, 1996, Proceedings., Fourth Workshop on, March 1996, pp. 240–247.
  - [49] M.A. Storey, *Theories, tools and research methods in program comprehension: past, present and future*, Software Quality Journal **14** (2006), no. 3, 187–208.
  - [50] E. Burton Swanson, *The dimensions of maintenance*, Proceedings of the 2nd international conference on Software engineering, IEEE Computer Society Press, 1976, pp. 492–497.
  - [51] P. Tonella, M. Torchiano, B. Du Bois, and T. Systä, *Empirical studies in reverse engineering: state of the art and future trends*, Empirical Software Engineering **12** (2007), no. 5, 551–571.
  - [52] W. M. Ulrich, *Knowledge Mining: Business Rule Extraction & Reuse*, System Transformation Portal, February 2009.
  - [53] X. Wang, J. Sun, X. Yang, Z. He, and S. Maddineni, *Application of information-flow relations algorithm on extracting business rules from legacy code*, Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on, vol. 4, IEEE, 2004, pp. 3055–3058.
  - [54] X. Wang, J. Sun, X. Yang, Z. He, and S. Maddineni, *Business rules extraction from large legacy systems*, Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on, March 2004, pp. 249–258.
  - [55] I. Warren and J. Ransom, *Renaissance: a method to support software system evolution*, Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International, IEEE, 2002, pp. 415–420.
  - [56] B. Xu, J. Qian, X. Zhang, Z. Wu, and L. Chen, *A brief survey of program slicing*, ACM SIGSOFT Software Engineering Notes **30** (2005), no. 2, 1–36.

- 
- [57] E. Yourdon, *Structured Analysis Wiki*, <http://yourdon.com/strucanalysis>, February 2011.
  - [58] A. Zaidman, T. Calders, S. Demeyer, and J. Paredaens, *Applying Webmining Techniques to Execution Traces to Support the Program comprehension Process*, Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on, IEEE, 2005, pp. 134–142.