# Recovering Business Rules from Legacy Source Code for System Modernization

Putrycz, Erik; Kark, Anatol

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC · CNRC

# *Recovering Business Rules from Legacy Source Code for System Modernization* *

Putrycz, E., Kark, A.
October 2007

* published in the Proceedings of The International RuleML Symposium on Rule Interchange and Applications (RuleML-2007). October 25-26, 2007. Orlando, Florida, USA. Lecture Notes in Computer Science. NRC 49855.



Canada

# Recovering Business Rules from Legacy Source Code for System Modernization

Erik Putrycz, Anatol W. Kark

Software Engineering Group, National Research Council Canada
{Erik.Putrycz, Anatol.Kark}@nrc-cnrc.gc.ca

**Abstract.** By using several reverse engineering tools and techniques, it is possible to extract business rules from the legacy source code that are easy to understand by the non-IT experts. These business rules can be used at different stages of system modernization. System maintainers can use the rules to locate in the code parts affected by a change in a rule. Business analysts can use those rules as means to aide understanding of the system at a business level. The extracted rules can serve as source of documentation and possible input for configuring a new system. This paper presents a novel approach for extracting business rules from legacy source code and application of the results at different stages of system modernization.

**Keywords:** Reverse engineering, Business Rules, Information Retrieval, System modernization.

## 1 Introduction

Governments and large corporations have a huge amount of the legacy software as part of their IT infrastructure. In 2006, 70% of all transaction systems were written in COBOL [1]. Understanding and discovery of business rules play a major role in the modernization of legacy software systems. According to a recent survey [2], about 51% of companies who have difficulties modernizing said that a major issue was the fact that "hard-coded and closed business rules" make it difficult to adapt their systems to new requirements and migrate to more modern environments.

This paper presents a process and the supporting tools for extracting business rules from legacy source code. Section 2 describes the context of legacy software system and its renovation, Section 3 details a process we used in analyzing and extracting business rules from COBOL code, while Section 4 presents and tool we developed and its possible use and early results of the analysis. Finally, Section 5 presents the conclusions and outlines direction of future research.

## 2 Legacy Software and system renovation

According to [3], "Legacy Information Systems" (LIS) have the following characteristics:

- Usually run on obsolete hardware that is slow and expensive to maintain.
- Software maintenance can also be expensive, because documentation and understanding of system details is often lacking and tracing faults is costly and time consuming.
- Lack of clean interfaces makes integrating LISs with other systems difficult.
- Are difficult, if not impossible, to extend.

We consider for this research large legacy "Enterprise applications" which were developed in-house and are critical to enterprise function. Those include pay, insurance, warehouse logistics and many similar systems.

The typical characteristics of these types of applications are:

- They handle and process a large amount of data.
- They have a large user base.
- They involve many business rules linked to legislation, processes and other legal matters.
- Many business rules are not accurate anymore and require external error and exception processing because of the difficulty to improve and update such system.
- The maintenance of the system often involves changes related to business rules.
- The source code is available.

By modernization, we mean here any process for evolving a system. This can be achieved by many different ways [4]. For example the legacy system can be replaced by a new one, or it can be interfaced with a new system. In any case, business rules in the legacy system play a major role. If the legacy system is being replaced, then the business rules are essential source of information for building the new system. If the legacy system is not phased out, then locating and analyzing the existing rules will help evolving and maintaining the legacy code.

In this context, we consider two main types of "business rules" [5]:

- From the business perspective, legislative rules, business practices and operational rules that are usually linked to natural language documents;
- From the Information System perspective, high level rules that are interpreted from the operational and legislative rules into executable rules that can be used in a system, possibly a business rules engine.

Two main classes of the stakeholders are involved with business rules:

- *Legacy system maintainers* fix bugs and implement new rules in the legacy system. They need to understand, not just the required changes, but the

business rules they are affecting and the execution paths to a specific business rule.
- *Business analysts* must understand a context in which new requirements of a system exist and the effect those requirements will have on the existing rules.

## 3 Extracting Business Rules from Legacy Code

In the context of a system modernization, our objective is to provide tools that extract business rules from the legacy code, to support both a legacy system and a new system construction. Consequently, the requirements for extraction tools are:

- All extracted artifacts have to be traceable to the source code;
- Business rules must be at high level and understandable by all stakeholders involved, including business analysts.

In this section, we will focus on analyzing and extracting business rules from the COBOL programming language in a way that satisfies our stated objectives. The process and tools presented in this paper have been implemented for COBOL and were developed as part of two large system modernization projects in which we are currently involved. While COBOL is our source language we believe that the same method can be applied to other programming languages.

### 3.1 Business rules definition

According to the Business Rules Group, *business rules* are, from the business perspective, an obligation concerning conduct, action, practice, or procedure within a particular activity or sphere. From the information system perspective, a business rule is a statement that defines or constrains some aspect of the business [5]. The Semantics of Business Vocabulary and Business Rules (SBVR) standard [6] defines a standard vocabulary and a process for writing business rules. We will use here a small subset of SBVR called "production business rules" [7]. They have the following syntax [8]:

<*conditions*> <*actions*>
where:
<*conditions*>
1. consists of one or more Boolean expressions joined by 'logical' operators ("and", "or", "not")
2. must evaluate to a "true" state for the rule's actions to be considered for execution;

and

<*actions*>
1. consists of one or more Action expressions
2. requires the rule conditions to be satisfied (evaluate to true) before executing.

### 3.2 Business rules extraction and system renovation

There has been lot of research done on system renovation techniques. Some of these techniques include [9]:

- reverse engineering,
- forward engineering,
- redocumentation,
- design recovery,
- restructuring,
- reengineering (or renovation).

According to [10], *reverse engineering* is concerned only with investigation of a system. *Redocumentation* is focused on making a semantically equivalent description at the same level of abstraction. Existing work on COBOL reverse engineering has focused on extracting diagrams [11], or translating it to other languages [12, 13]. A product called HotRod from Netron [13] aims to extract business rules from COBOL code. The granularity and extraction process are proprietary, but a user can define how many statements constitute a business rule. Additionally, to get more accurate results for each system, the algorithm used by HotRod should be tuned with the assistance of a COBOL expert. Once the tool has extracted business rules, a user needs to review the results and decide what do to; either use them for documentation purposes or use the results to restructure the code. If the results are to be used for documentation purpose, the results can be exported into a different documentation tool.

A product from Evolveware called S2T helps to automate the redocumentation process [14, 15] of legacy software. The tools analyses the source code using a knowledge engine that is "trained" for data discovery and knowledge extraction. The extracted information can also be an input for a new system. The business rules are focused on extracting the execution flow and can analyze end user input/output flows. In the field of software maintenance, related work on redocumentating legacy systems has been done. In [16], the authors present a formal ontological representation that integrates Java source code and software documentation. Their tool parses Java code and using the AST they populate ontologies into a database. Using a custom natural language processor, they construct a second set of ontologies and then link it to the source code set. This process reduces the work required for redocumenting a system.

### 3.3 From COBOL to business rules

The structures of legacy COBOL and today's object oriented programs are radically different. COBOL has a limited and simple structure - each program contains a sequence of statements grouped into *paragraphs* which are executed sequentially. COBOL has only two forms of *branching*; one to execute external programs (CALL) and another to transfer the control flow to a paragraph (PERFORM). PERFORM is also used for iterations. Each program is associated with a single source file.

To extract useful information from the source code, it is necessary to find heuristics that separate setups, data transfers (usually from flat files) from business processing. To separate those two aspects, we focus on single statements that carry a business meaning such as calculations or branching since they most often represent high level processing.

In the case of branching (as defined above), Paragraph names and external programs can either be traced to documentation or the names themselves could possibly be self-explanatory. In code we have investigated, we found large number of paragraph names such as "CALCULATE-BASIC-PAY" which can be understood by a business analyst.

To make calculations easier to understand, it is necessary to translate the variable names, which more often are abbreviations or codes, into non-technical (their semantical) terms. We have encountered a large number of variable names such as CC35-CUR-APR or WS-PREV3-SUPN-LOW-MAX that require translation to the business terms.

Once calculations and branching are located, it is necessary to construct their *context*. Here, by context we mean all conditions in which a calculation or branching operation happens. This includes all IF statements under which an operation might be executed and also iteration information. This context also needs to be translated into non-technical terms. The conditions and statements are then converted into production business rules.

### 3.4 Knowledge Extraction process

This business rules construction process is based on abstract syntax tree (AST) analysis. An abstract syntax tree is a tree, where the internal nodes are labeled by operators, and the leaf nodes represent the operands of the operators. This tree is obtained by parsing the source code. The parser required to construct the AST for this task had to be specifically designed to analyze and include all elements of the source code including comments. Many existing parsers are designed for other purposes such as syntax verification or transformation and they did not suit our objectives.

After creation of the AST, the business rules extraction is divided into two steps (Figure 1). First, we analyze the AST and construct a knowledge database. Once that database is constructed, we simplify the data collected (detailed in Section 4.3) and link the artifacts, wherever possible, to the existing documentation (Section 4.1). This is done by adding various semantic information, usually links connecting variable references, to the corresponding declarations.

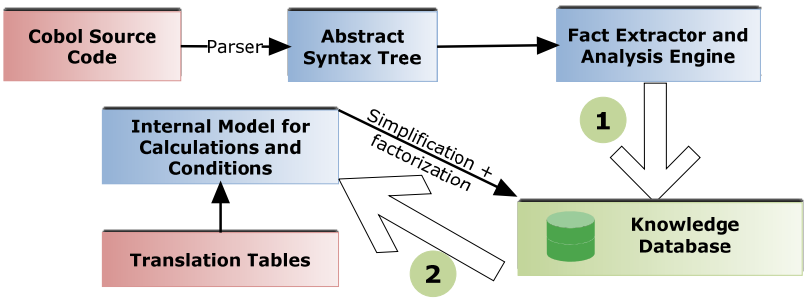


Figure 1: Source code analysis process

The knowledge elements extracted from the abstract syntax tree are represented in Figure 2. The extracted elements are:

- production business rules: if *condition*, do *action*;
- conditions: one or more Boolean expressions with variables and constants joined by 'logical' operators;
- business rules: an action, possibly a condition and a comment from the source code;
- actions can be either branching statements or calculations with identifiers and constants;
- identifiers: variables used in conditions and calculations;
- Code blocks: they represent one or more lines of code in the original program;
- business rules dependencies: some calculations in other business rules executed before may affect the current calculation thus a business rules can be linked to one or other ones;
- loop and branching information: the paragraph in which the business rule is located may be called in a loop from another location in the program;

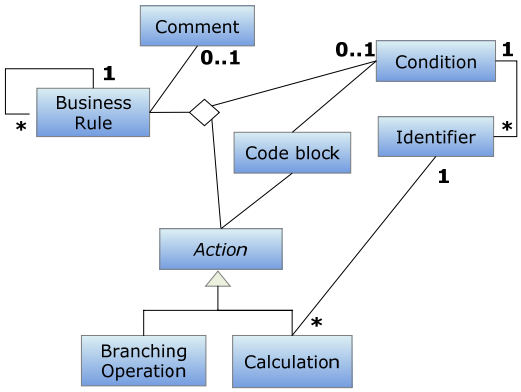


Figure 2: UML Class diagram of the knowledge database

The following example illustrates the business rules extracted from a fragment of the COBOL code. It is important to notice that the *IF* conditions get unfolded and the two COMPUTE statements are treated as separate (not part of one IF statement).The extracted rules in this example have not yet been subjected to the semantical analysis.

COBOL code:

```
019872       IF M39PEN-CD = 18
019873          IF FUND2-PREV-YR-202 > 0
019874             COMPUTE FUND2-5B7-202 =
019875                     FUND2-5B7-202
019876                   + (FUND2-GROSS-202
019877                    * CORRCTNL-FACTOR)
019878          END-IF
019879          IF FUND3-2PYR-GROSS-202 > 0
019880             COMPUTE FUND3-5B7-202 =
019881                     FUND3-5B7-202
019882                   + (FUND3-GROSS-202
019883                    * CORRCTNL-FACTOR)
019884          END-IF
019885       END-IF
```

And extracted rules:

- BR-1: *if* M39PEN-CD = 18 and FUND2-PREV-YR-202 > 0, *calculate* FUND2-5B7-202 = FUND2-5B7-202 + (FUND2-GROSS-202 * CORRCTNL-FACTOR)

- BR-2: *if* M39PEN-CD = 18 and FUND3-2PYR-GROSS-202 > 0, *calculate* FUND3-5B7-202 = FUND3-5B7-202 + (FUND3-GROSS-202 * CORRCTNL-FACTOR)

## 4 Structuring and navigating extracted business rules

In order to enable a business analyst to understand the rules collected, all technical terms have to be translated into business terms. We are using several mechanisms for achieving this goal:

- Dedicated user interface for visualizing and navigating the collected data
- Integrating documentation
- Simplification of conditions in business rules

### 4.1 Business rules visualization

As a part of our work, we have built a rules visualization and browsing tool. Additionally, the tool can generate dynamically an execution graph for each of the rules. The tool takes as input the results of the source code analysis – the knowledge database.

The current implementation of the business rules extraction tools has been tested on 18 COBOL programs of a large legacy system representing about 630,000 lines of code. Our extraction tool found 9199 business rules, 4825 of them are calculation and 4374 branching operations. The complete generation of the knowledge database requires about one hour for analyzing the source code and generating business rules. However, the generation time has not been an issue since the database doesn't need to be updated instantly.

Figure 3 shows the main elements of the graphical user interface of the tool. The left panel (item 1) is used to navigate the business rules in different ways. The right panel (items 2 and 3) displays the information about a selected element in the left panel. In this case, Figure 3 displays the details of a business rule. Those details include the context of the rule (current paragraph and current program in item 2), conditions and the action itself (item 3). All variable names are translated using the strategy detailed later. Using the type information, we color each element type (identifiers, constants, etc.) differently. When the action is a calculation, we convert it to an equation and render it in a form that business analysts are more used to.

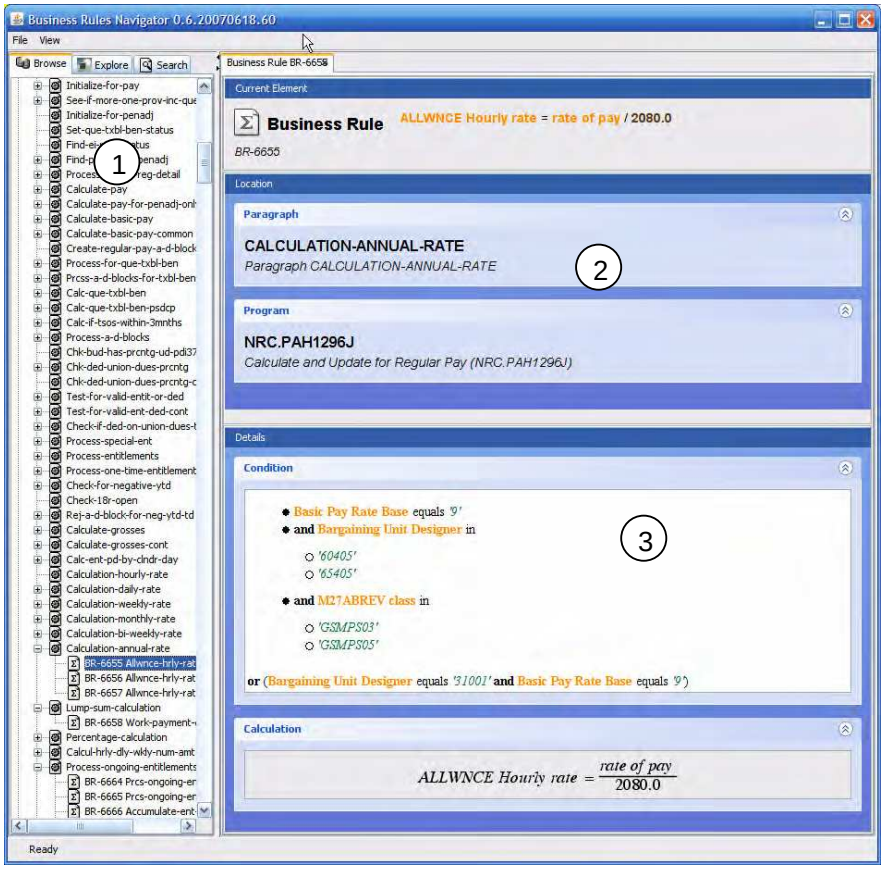


Figure 3 Business Rule visualization

In the knowledge database, the conditions and actions of business rules are associated with *code blocks* (Figure 2). A code block can be any (disjoint) section of the source code. Figure 4 shows highlighting used to trace the conditions and actions of a business rule to the source code. When the user selects a business rule and opens the source code view tab of the tool, the code sections corresponding to the conditions and the action are highlighted. This feature is targeted at system maintainers who need to update and modify the business rules.

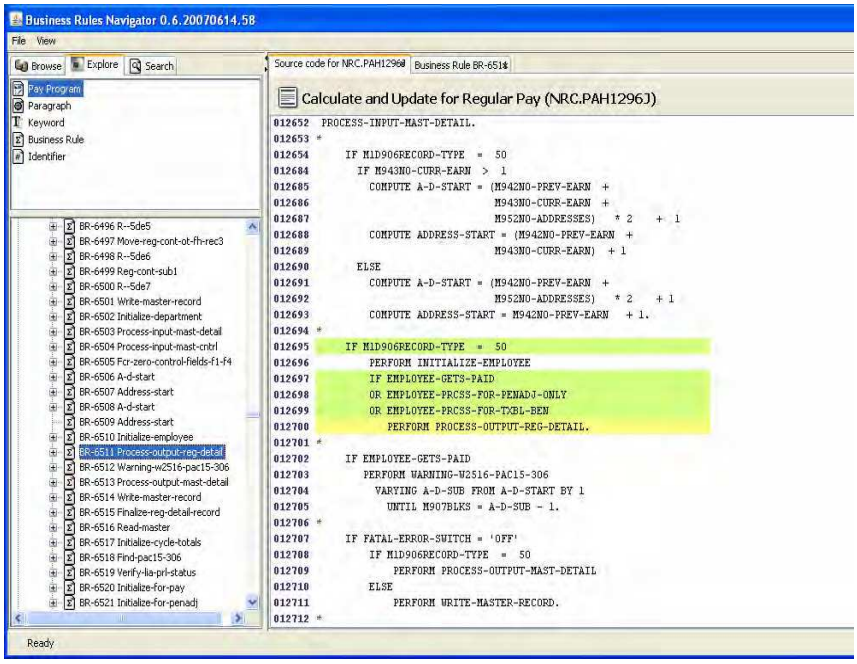


Figure 4: Business rules and source code linking

### 4.2 Integrating documentation

Although legacy systems are often not well documented, there usually is some documentation of the data processed, including at least the names of the fields and their descriptions. Our objective is to link that existing documentation with the business rules.

One way to achieve this is through variables names. For each variable, we try different strategies to translate their technical names. At first, if the variable is directly linked to a data field (usually a field of the so called “master file”) then we use that field description. Otherwise, we look in the AST and search for variable assignments and copies that can link one variable to a documented one. This scenario happens often in COBOL because the developers use temporary variables to do all processing after reading and before writing the data. Finally, if we cannot trace the variable to a documented field, we use a translation table with abbreviations and their common meaning. This process ensures that the majority of identifiers are presented with their business semantics.

### 4.3 Simplification of conditions in business rules

The condition of a business rule is based on all of the *IF* statements which have to be true for an action to be executed. The sum of all these conditions can easily result in a long and complex expression difficult to understand. To greatly reduce the size and complexity of the expressions, we use - according to the user's preferences - either a conjunctive normal form (CNF) or a disjunctive normal form (DNF).
In addition to the normal form, we use type inference to match constants used in the expressions with data tables. A custom type can be assigned to variables in our knowledge database which then can be used by the expression builder to assign these types to constants. The user interface translates these constants using custom types and existing data files or database tables.

For instance the following condition "ELEMENT62-ENT-DED-CD = '304' OR '305'" can be translated to "Entitlement Deduction in (overpayment, rec o/p extra duty)". By inferring the type "deduction" to '304' and '305', their meaning can be found in an external data table.

### 4.4 Navigation modes

Recent studies [17] show how that a flexible navigation is a key element for software maintainers. According to their findings, the development tools have to show dependencies, allow navigation among them and provide an easy way to judge of the relevance of information. It is also necessary to provide means to collect the relevant information.

For these purposes, three different types of navigation of the extracted information are available:

- Navigation in a tree with the following hierarchy: the different COBOL programs, the paragraphs and the business rules inside each paragraph (Figure 5);
- Navigation through the whole system using keywords attached to business rules, identifiers, paragraphs and comments; Figure 6 displays an example of all business rules with the keyword "TAX" attached to them.
- Custom navigation using tags: users can create and add tags to any element and then navigate by tag through these elements.

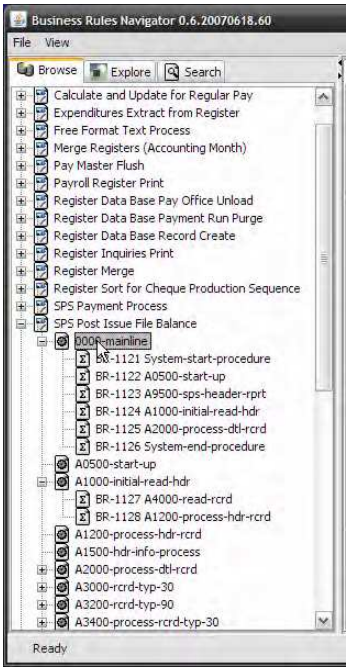


Figure 5: Basic navigation

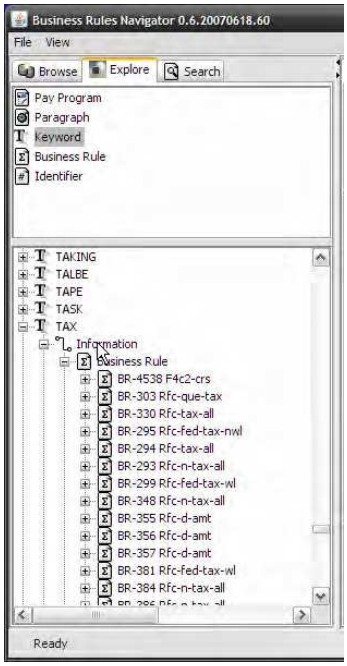


Figure 6: System wide navigation

### 4.5 Execution graph visualization

The objective of the execution graph visualization is to help understanding in which context one business rule might be executed. By context, we mean here all the executions paths in a program that can lead to the execution of the rule. Figure 7 shows an example of the different execution paths for the business rule BR-6501. The rule is at the bottom of this example graph. The remaining nodes in the graph are paragraphs preceding the execution of BR-6501. The edges represent the ordering. A label on the edge represents a condition under which the path is taken.

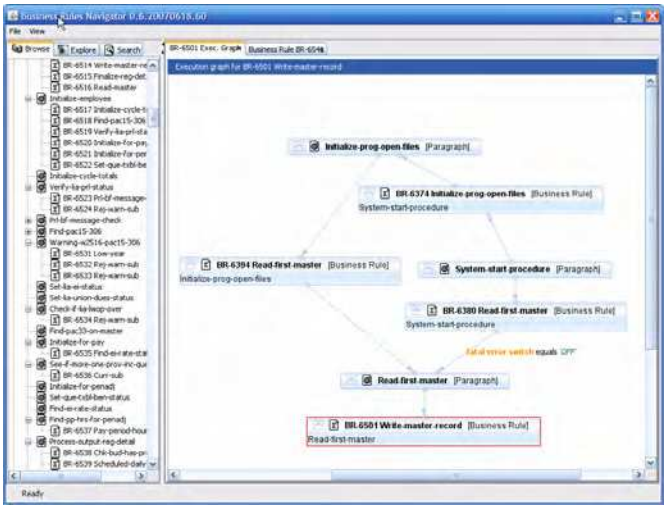


Figure 7: Execution graph visualization

## 5 Conclusions

This paper presents means of helping a modernization process by extracting business rules from legacy source code. The novelty in this research is that the output is targeted at business analysts. This means that the business rules must be translated into non-technical terms. The extraction of rules from source code is divided into two main steps. First, the source code is parsed into an abstract syntax tree to locate the calculations and other elements of business rules. Second, a context is constructed for this information to transform it into a form which is easier to understand. This is done by analyzing all the conditions of a business rule and translating them to non-technical terms. Given the large amount of information collected, we implemented a dedicated interface to navigate the business rules. This process and tools are currently being used in two large projects for both modernization and legacy code maintenance. Early feedback from business analysts has been extremely positive about the value and understandability of the information extracted. We plan to evaluate the effectiveness of the business rules extraction process by studying how business analysts use the results. In addition, we are looking at integrating natural language techniques such as co-location for improving the translation of identifiers.

## 6 References

[1] Krill, P.: The future's bright ... the future's COBOL (2006)
[2] Software AG: Customer survey report: Legacy modernization. Technical report (2007)

[3] Bisbal, J., Lawless, D., Wu, B., Grimson, J.: Legacy information systems: issues and directions. Software, IEEE **16**(5) (Sept.-Oct. 1999) 103–111
[4] Koskinen, J., Ahonen, J., Sivula, H., Tilus, T., Lintinen, H., Kankaanpaa, I.: Software modernization decision criteria: An empirical study. In: Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on. (21-23 March 2005) 324–331
[5] Business Rules Group: What is a business rule? http://www.businessrulesgroup.org/defnbrg.shtml (2007)
[6] OMG: Semantics of Business Vocabulary and Business Rules (SBVR)
[7] Hendryx, S.: SBVR and MDA: Architecture. Business Rules Journal **6**(11) (nov 2005)
[8] OMG: Production Rule Representation Request For Proposal. (2003)
[9] Chikofsky, E.J., II, J.H.C.: Reverse engineering and design recovery: A taxonomy. IEEE Software (January 1990) 13–17
[10] Brand, M.G.J. van den, P.K., Verhoef, C.: Reverse engineering and system renovation: an annotated bibliography. ACM Software Engineering Notes **22**(1) (January 1997) 42–57
[11] Edwards, H.M., Munro, M.: RECAST: reverse engineering from COBOL to SSADM specification. (April 1993) 499–508
[12] Gray, R., Bickmore, T., Williams, S.: Reengineering Cobol systems to Ada. Technical report, InVision Software Reengineering, Software Technology Center, Lockheed Palo Alto Laboratories (1995)
[13] Arranga, E.: COBOL tools: overview and taxonomy. Software, IEEE **17**(2) (March-April 2000) 59–69
[14] EvolveWare Corporation: S2T legacy documentation tools. http://www.evolveware.com (2007)
[15] EvolveWare Corporation: Evolveware's S2T technology, a detailed overview. http://www.evolveware.com (2007)
[16] Witte, R., Zhang, Y., Rilling, J.: Empowering software maintainers with semantic web technologies. In: Proceedings of the 4th European Semantic Web Conference. (2007)
[17] Ko, A., Myers, B., Coblenz, M., Aung, H.: An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. Software Engineering, IEEE Transactions on **32**(12) (Dec. 2006) 971–987