Connecting legacy code, business rules and documentation

Putrycz, Erik; Kark, Anatol

Publisher's version / Version de l'éditeur:

The 2008 International RuleML Symposium on Rule Interchange and Applications (RuleML 2008) [Proceedings], 2008

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC · CNRC

Connecting Legacy Code, Business Rules and Documentation *

Putrycz, E., Kark, A.
October 2008

* published in The 2008 International RuleML Symposium on Rule Interchange and Applications (RuleML 2008). October 30-31, 2008. Orlando, Florida, USA. Lecture Notes in Computer Science. NRC 50413.

Canada

Connecting legacy code, business rules and documentation

Erik Putrycz and Anatol W. Kark

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 legacy source code that are easy to understand by the non-IT experts. To make this information usable to business analysts, it is necessary to connect the artifacts extracted to existing documents. In this paper, we present how we use source code analysis and keyphrase extraction techniques to connect legacy code, business rules and documentation.

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

1 Introduction

Governments and large corporations maintain huge amount of the legacy software as part of their IT infrastructure. In 2008, 490 companies of the Fortune 500 are still using legacy systems to process more than 30 billion transactions or $1 trillion worth of business each and every day. In Canada, a recent report [1] estimates that 60,000 employees - which is 10% of the 600,000 total ICT employment - are working with legacy systems.

Understanding and discovery of business rules play a major role in the maintenance and modernization of legacy software systems. The recovery of business rules supports legacy asset preservation, business model optimization and forward engineering [1]. According to a recent survey [2], about half of the companies who reported difficulties in modernizing their legacy systems, 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. In our previous work [3], we described how we extract business rules from legacy code. However, business analysts need more than just the artifacts from code. In this paper, we present how we use HTML extraction techniques and keyphrase analysis to link the source code artifacts to their technical and other related documents. We also describe an intuitive navigation tool designed for the use by the business analysts. This paper presents the results of analyzing approximately 700,000 lines of the COBOL source code and 4000 HTML and Microsoft Word documents.

2 Background

In this paper, we focus on large legacy systems. These systems share the following characteristics [4]:

- Often run on obsolete hardware that is slow and expensive to maintain.
- Are expensive to maintain, because documentation and understanding of system details is often lacking and tracing faults is costly and time consuming.
- Lack of clean interfaces make the integration with other systems difficult.
- Are difficult, if not impossible, to extend.

These systems are found in most sectors such as pay, insurance, warehouse logistics and many other sectors.

Several factors motivate industry and governments to migrate from their legacy systems [1]:

- The hardware is no longer supported (e.g. chip sets become obsolete);
- The system becomes error prone;
- The system or applications no longer fit the business;
- Key in-house people retire or leave and replacements are difficult to recruit and/or train (including contractors);
- There is a herd mentality in some industries toward a new technology solution;
- Senior management becomes concerned about the risks involved.

These legacy systems often date from 1970s when the concepts of proper software engineering were relatively new and proper documentation techniques were not a concern. Old COBOL applications are typically monolithic, with hundreds of thousands of lines of code mixing the business logic, user interface, and transaction data in no particular sequence [5]. As a consequence, migrating from these systems is a complex and expensive process. When possible, these legacy system are kept intact, wrapped and integrated in a more modern system. In other cases, it is necessary to re-engineer the whole system.

Without the knowledge from the legacy code, analysts would need to rewrite all the business rules using all the legislations, policies and other agreements. These rules usually consists of calculations, exceptions and other elements that can be recovered from the code.

In our previous publication [3], we detailed how we extract business rules from legacy COBOL code and provide a basic linear navigation. The feedback from analysts showed us that this linear navigation is not sufficient and they need to locate business rules for a specific topic or an existing document.

In the rest of the paper, we detail how we achieved this objective. First, we present two existing solutions and detail why they are not appropriate for our case (Section 3). Then, we present our approach, which consists of first extracting business rules and other artifacts from source code and building a knowledge database (Section 4). Once that is built, we link the identifiers used

in business rules to existing technical documents (Section 5). Using these links and keyphrase extraction techniques, we are able to connect the business rules extracted from the source code to many other documents (Section 6). In Section 7 we present short conclusions and outline further work.

3 Related work

Several approaches exist to connect documents to source code. Witte et al. [6, 7] propose an approach to extract semantic information from source code and documents and populate an ontology using the results. They use native language processing to analyze technical documentation and identify code elements in the documentation. To connect the documents together, they use a fuzzy set theory-based coreference resolution system for grouping entities. Rule-based relation detection is used to relate the entities detected. The text mining approach achieves 90% precision in the named entities detection. However after adding the source code analysis, the precision drops to 67%.

In our work, we use only data description documents. Since the those documents were not by and large "written in prose" the information contained these documents had to be extracted using different techniques from natural language processing. Additionally, our approach is centered on the code and documents used to support artifacts extracted from the code.

In [8], Antoniol et al. look at the problem of recovering traceability links between the source code of a system and its free text documentation. They use information retrieval techniques to trace C++ code to free text documents. The process normalizes identifiers from the source code and does a morphological analysis of the software documents. Both are indexed and a classifier connects them. Unfortunately the precision achieved isn't very high.

In our case, the documentation considered is quite different. We focus our work on the application specific data instead of the programming interface. The data considered do have complex names that can be found accurately with full text search. Thus, natural language analysis is not necessary to locate identifiers. In addition, since the documents considered share a similar structure, it is possible to locate specific parts of the documents their formatting.

4 Extracting knowledge from source code

In the context of a system modernization, our objective is to provide tools that extract business rules from the source code, to support both a legacy system maintenance and 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 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 current source language, we believe that the same method can be applied to other programming languages.

4.1 Business rules definition

We use a small subset of SBVR [9] called "production business rules". They have the following syntax:

$< conditions >< actions >$

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 rules 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.

4.2 From COBOL to business rules

The structures of legacy COBOL and todays 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 and 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 be either traced to documentation or the names themselves can 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.

Once calculations and branching are located, it is necessary to construct their context. 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 plus iteration information. We consider two contexts for one operation

– local context: the IF statement surrounding the operation;
– global context: all the possible IF statements - starting from the beginning of execution - that can lead to the operation.

Depending on the operation and on the information required, either the local or the global context might be the most appropriate.

4.3 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.

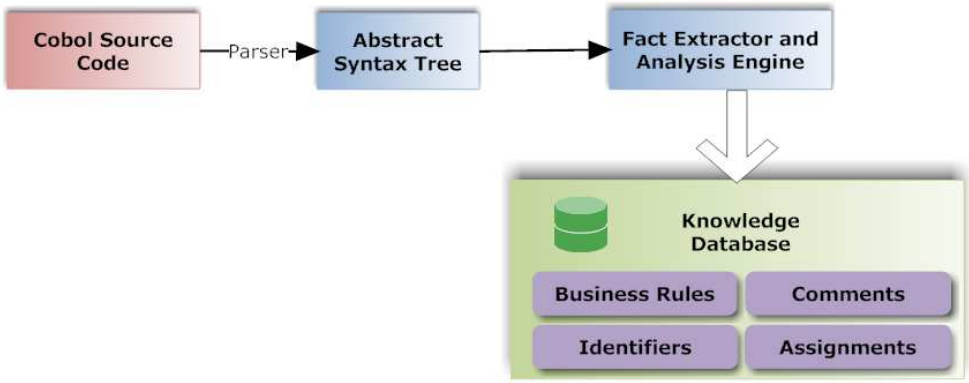

Fig. 1. Extraction step 1: Source code analysis

The business rules extraction is divided into two steps. (Figure 1). First, we analyze the AST and construct a knowledge database. Second, once that database is constructed, we simplify the data collected and link the artifacts, wherever possible, to the existing documentation (Section 6).

The extracted elements of the knowledge database 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; where action can either be branching statement or calculations with identifiers and constants;

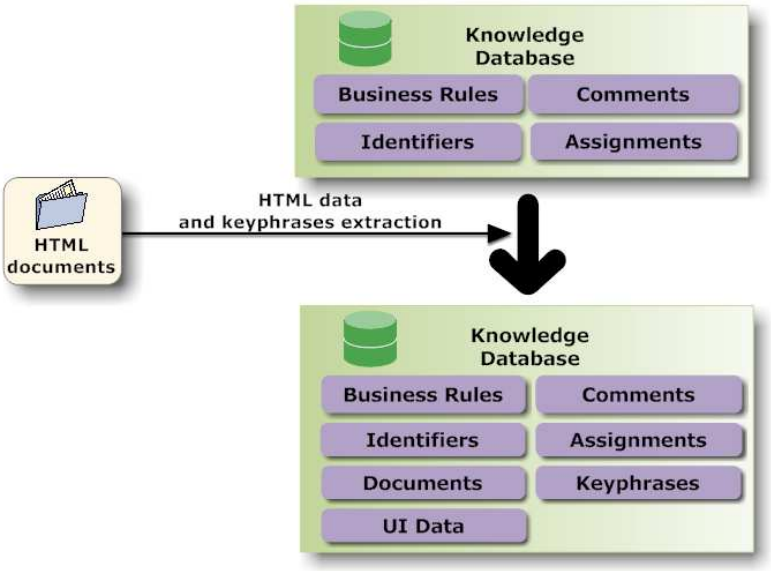

Fig. 2. Extraction step 2: Documentation integration

- 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;
- exceptions: they are all operations leading to an error message or to an aborted execution.

The second step (Figure 2), connects the documentation to the artifacts previously extracted (detailed in Section 6). All documents are first converted to HTML format and then keyphrases are extracted. All this information is added to the database. In addition, all the navigation indexes used on the end-user graphical interface are calculated and added to the database.

4.4 Identifying temporary identifiers

A common pattern found in legacy programs is the usage of temporary identifiers for performing a set of operations, instead of the identifiers attached to the data fields. An example is presented in Figure 3. To link the identifiers to a data field in many calculations, it is necessary to track assignments and other operations that copy the value of one identifier into another. Tracing these temporary identifiers can be very valuable for documenting the operations being performed. When the

identifers of operation are documented through temporary identifers, we call a *transitive connections* the connection between the document and the identifiers. This connection is not totally accurate because we cannot verify the order of the assignations through the execution. The accuracy of the translations is discussed in Section 5.2.

```
Load identifier A from database
Temporary identifier Ta = A
...
Calculate Ta = ...
...
Set A = Ta
Save A in database
```

Fig. 3. Temporary identifier usage example

5 Linking identifiers to technical documents

In our previous work [3], we used a translation table and a list of abbreviations to convert an identifier to a meaningful name. However, that approach did not provide satisfactory results with a different set of COBOL programs. In addition to this issue, the accuracy of the results obtained with the translation table and abbreviations was dependent on the accuracy of the translation tables themselves. Those tables were also "legacy" and were proven not well maintained.

5.1 Translating identifiers

To translate identifiers to business names, we decided to rely on an existing documentation of the data. The documentation describes all data fields used in programs and in databases. An example of a document is showed in Figure 4. (The actual data was obfuscated.)

The document details a data element and its use across all the systems. All these documents have been generated by technical writers and they all share a similar structure with sections and fields.

We make use of this similarity to extract information. As first step, all documents are normalized to a hierarchical structure. In our knowledge database, documents have the following structure:

- Document group: contains a set of documents;
- Document: contains a sequence of sections;
- Section: contains a sequence of paragraphs.

Axx Indicator

Technical Name:AXX_YYY_IND
Definition: AXX YYZZs Indicator within YYZZs Codes Control File
Model Status:System Information/Skip: Reviewed during conceptual data modeling process
- bypassed because this is a current computer system/application specific term
and may not be required
for a system rewrite/business re-engineering.
Indicates whether a particular YYZZs can appear in an AXX transaction (deduction and YYZZs form).
System(s): System1
System2
System3
Element Type: Business
Data Class: Indicator
Data Type: Base
Data Structure:1 character, alphanumeric

System1
Notes: Synonym is: OL-YYZZ-AXX
V1-YYZZ-AXX
V2-YYZZ-AXX
Valid Values:Y, N
Input Forms:N/A
Element Name:YYZZ-AXX
- subordinate to:GO-YYZZ
GO**-**YYZZSES
GO**-**DATA
GOSS
Picture:PIC X(01)
Subordinate
Elements:N/A
File ID/Records
Description
MM200-XXXX-YYYY-LR logical record used by input/output module
MM401-SB-XXXX-YYYY-MMMMMM logical record used to build online screen

System2
Notes: Synonym is: OL-YYZZ-AXX
V1-YYZZ-AXX
V2-YYZZ-AXX
Valid Values:Y, N
Input Forms:N/A
Element Name:YYZZ-AXX
- subordinate to:GO-YYZZ
GO**-**YYZZSES
GO**-**DATA
GOSS
Picture:PIC X(01)
Subordinate
Elements:N/A
File ID/RecordsDescription
MM200-AAAA-BBBB-LR logical record used by input/output module
FF401-XX-YYYY-ZZZZ-UUUUUU logical record used to build online screen

System3
Notes: Synonym is: OL-YYZZ-AXX
V1-YYZZ-AXX
V2-YYZZ-AXX
Valid Values:Y, N
Input Forms:N/A
Element Name:YYZZ-AXX
- subordinate to:GO-YYZZ
GO**-**YYZZSES
GO**-**DATA
GOSS
Picture:PIC X(01)
Subordinate
Elements:N/A
File ID/Records Description
MM200-ZZXX-MMNN-LR logical record used by input/output module
MM401-ASDFG database record
MM401-AA-BBBB-CCC logical record used to build online screen

Fig. 4. Sample technical document

In order to use the documentation to translate a COBOL identifier into a term understood by business analysts , it is necessary to analyze the document, locate the identifier and the context in which it is found. Locating the identifier in the documents has been achieved using a customized full text search engine. Full text search engines usually tokenize all text using a pre-defined set of separators. We had to implement a specific tokenizer that filters identifiers in the documents and leaves them intact in the index. In the documentation, inside each section, are a list of field definitions in the form "*field: value*" . Once an identifier is located in a section, we verify that it is located in a field value and that the section name and the field name are valid . Locating valid positions for identifiers is done in three steps

1. Extract all field names using the HTML markup and position in HTML tree;
2. Consolidate and normalize the extracted field names and remove false positives (because the documents are hand written, several variants exist for each field name and the HTML markup is not used consistently);
3. Re-parse all the HTML documents with the normalized field names and extract correct sections and fields.

If the location is valid, the title of document is used as translation of the technical name. For instance, when looking for a translation of *MM200-ZZXX-MMNN-LR* in Figure 4, we locate it in section "System 3" and in the field "File ID/Records Description".

Despite the fact that legacy system are usually poorly documented, we expect that similar documents such as the ones used in this case and presented in Figure 4 can be found. A technical documentation on a system is not vital for running an IT system, however understanding the data processed is necessary.

5.2 Accuracy of translations

With the process described above, when translating an identifier found only once and in a valid location, we can assume the translation is accurate. When an identifier is found in multiple documents, we introduce for each identifier translation an accuracy factor called tr_a. tr_a is calculated as: $tr_a = 1/n_{documents}$. In addition, when a transitive connection is involved, because different execution paths can lead to different results, we decided to decrease the original accuracy. Currently we use 70% of the original accuracy. It is an arbitrary number used to decrease the accuracy when transitive connections are involved, this number has to be tuned and balanced with the original calculation of tr_a.

To achieve the best accuracy, our implementation calculates all possible translations and uses the one with the highest accuracy. In addition, on the user interface, we offer a configuration setting to specify the accuracy level in order to eliminate long transitive connection paths.

5.3 Extracting connections between documents

Once the identifiers have been linked to data documentation, it is possible to link the documents to business rules. To link a document to a business rule, we

locate all identifiers that are referenced in a document. Then, we search for all business rules that use one of the identifiers either as condition or as calculation and we link them to the document.

This offers a first level of navigation for locating any operation in a program connected to a data field.

To help analysts build a data flow in a new system and verify their results, we can extract dependencies between documents based on the business rules through a whole system. We consider two documents connected when at least one identifier found in each document appears in a business rule.

Figure 5 shows relations extracted from one document (called "AWW Quantity") and shows all other data sources involving *AWW Quantity* in an operation. A connector between two documents means that they are connected through identifiers in the business rule. The label on each connector is a business rule. The details on *"AWW Quantity"* are all programs and all identifiers connected to this documents and used in business rules.

6 Connecting external documents to code with keyphrases

In the previous section, we built a navigation with identifiers and documents. However, this navigation is still at a low level, and a higher level navigation with business rules and additional text documents is necessary. A typical scenario is to search for all business rules related to an external text document.

Using the identifier translations established previously, we are able to show all business rules related to a data field and its documentation. To improve the navigation, we decided to use keyphrase extraction to index all documents including the data documentation (that is being used for translating identifiers). This provides another benefit, an additional navigation with keyphrases and topics.

Indexing documents with keyphrases is done using a technique called *keyphrase extraction* detailed below.

6.1 What is keyphrase extraction

According to [10, 11], a keyphrase list is a short list of phrases (typically five to fifteen noun phrases) that capture the main topics discussed in a given document. They are often referred as keywords.

There are two main algorithms for keyword extraction, Kea [12, 13] and *extractor* [10, 11]. Both algorithms share the same basic techniques and measures, but Kea is based on naive Bayes whereas Extractor relies on a genetic algorithm - GenEx, adapted for its purposes. For our work we used Kea to validate the usage of keyphrase extraction because of its easy to use interface.

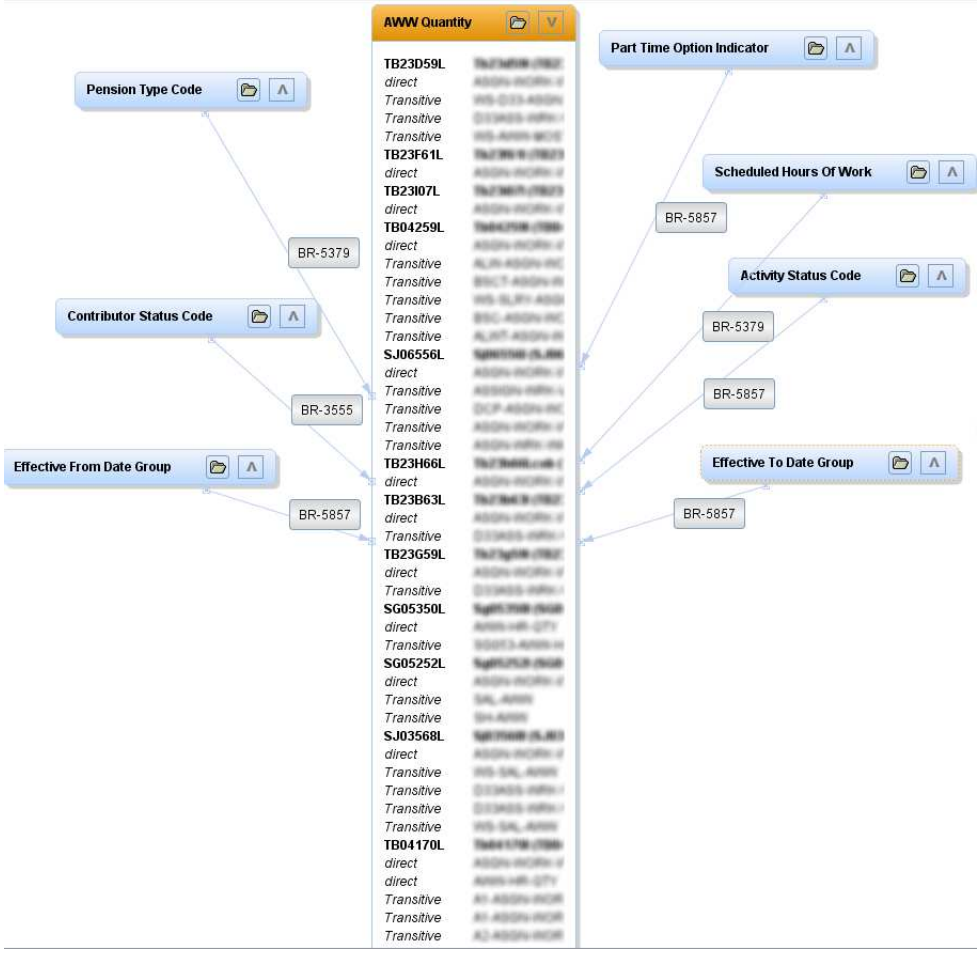

Fig. 5. Example of relations betweens documents

6.2 Kea

For each candidate phrase Kea computes four feature values:

- **TFxIDF** is a measure describing the specificity of a term for this document under consideration, compared to all other documents in the corpus. Candidate phrases that have high TFxIDF value are more likely to be keyphrases.
- **First occurrence** is computed as the percentage of the document preceeding the first occurrence of the term in the document. Terms that tend to appear at the start or at the end of a document are more likely to be keyphrases.
- **Length** of a phrase is the number of its component words. Two-word phrases are usually preferred by human indexers.
- **Node degree** of a candidate phrase is the number of phrases in the candidate set that are semantically related to this phrase. This is computed with the help of the thesaurus. Phrases with high degree are more likely to be keyphrases.

Kea comes out of the box without any training data, so it is necessary to provide a training data containing documents and manually extracted keyword so that it can build a model. The model contains the four feature values described in the previous section.

We used for training a set of 341 various types documents with manually specified keyphrases. A similar set has been used for *extractor* and has been proven sufficient to train the model for any kind of document. Given the feature values used, the model generated is then generic enough for any domain.

6.3 Results

To evaluate the grouping of document with keyphrase extraction, we used two set of documents. The first set of documents are the data field documents used previously to translate identifiers (detailed in Section 5). This set consists of 3603 files. The second set consists of 352 files which are documents frequently used by business analysts to lookup information on business rules.

Results of keyphrase extraction are presented in table 1. The keyphrase extraction tools work on a per document basis, so the keywords extracted are not consistent through the whole set. A total of 5608 unique keywords has been found. 4106 keywords (73%) have only one document matched in each set and thus are not useful for grouping documents.

To connect other documents with source code, we evaluated how many documents from set 1 are connected with the set 2. We found out that 329 documents in the set 1 (93%) are connected with 1941 in set 2 (53%) through 156 keyphrases. These connections may be sufficient and may require manual work to improve the accuracy but without any additional semantic information on the documents, the possibilities are limited.

The keyphrases found have not yet been validated by business analysts, however since most of them appear in their glossary, we consider them relevant.

Total number of documents in set 1	352
Total number of documents in set 2	3603
Total number of keyphrases	5608
Number of keyphrases with at least one document matched in each set	156
Number of keyphrases with at least two documents matched in set 1	207
Number of keyphrases with at least two documents matched in set 2	1427
Number of keyphrases with only one document matched in set 1	359
Number of keyphrases with only one document matched in set 1	3771

Table 1. Keyphrase extraction statistics

7 Conclusions

This paper presents means of helping a modernization process by extracting business rules from legacy source code and connecting the business rules to existing documents. The novelty in this research is that reverse engineering, information extraction and natural text processing are used to connect the code to documents. This enables to translate business rules into non-technical terms and helps business analysts to locate business rules using existing documents. 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. We translate identifiers using data documentation and by analyzing the usage of temporary identifiers. Once documentation is linked to identifiers, keyphrase extraction techniques enables to connect external documents.

This process and tools are currently being used in one large modernization project where business rules play a critical role. Feedback from business analysts has been extremely positive about the value and understandability of the information extracted. We are currently planning to improve document analysis to add more sources of documentation for identifiers. Also, we are looking at improving the context of the business rules and simplifying the results found.

Acknowledgements

We would like to thank to Peter Turney for all his help on keyphrase extraction.

References

1. Doyletech Corporation, Senik, D., Associates Inc.: Legacy applications trend report. Technical report, Information and Communications Technology Council (May 2008)
2. Software AG: Customer survey report: Legacy modernization. Technical report (2007)

3. Putrycz, E., Kark, A.: Recovering business rules from legacy source code for system modernization. In: Proceedings of RuleML 2007. (2007) 107–118
4. Bisbal, J., Lawless, D., Wu, B., Grimson, J.: Legacy information systems: issues and directions. Software, IEEE **16**(5) (Sept.-Oct. 1999) 103–111
5. Ricadela, A., Babcock, C.: Taming the beast. InformationWeek (2003)
6. Witte, R., Li, Q., Zhang, Y., Rilling, J.: Ontological text mining of software documents. **4592** (June 27–29 2007) 168–180
7. Witte, R., Zhang, Y., Rilling, J.: Empowering software maintainers with semantic web technologies. In: Proceedings of the 4th European Semantic Web Conference. (2007)
8. Antoniol, G., Antoniol, G., Canfora, G., Casazza, G., De Lucia, A.: Information retrieval models for recovering traceability links between code and documentation. In Canfora, G., ed.: Proc. International Conference on Software Maintenance. (2000) 40–49
9. OMG: Semantics of Business Vocabulary and Business Rules (SBVR). dtc/06-08-05.
10. Turney, P.D.: Learning algorithms for keyphrase extraction. Information Retrieval (2) (2000) 303–336
11. Turney, P.: Learning to extract keyphrases from text (1999)
12. Frank, E., Paynter, G.W., Witten, I.H., Gutwin, C., Nevill-manning, C.G.: Domain-specific keyphrase extraction. (1999)
13. Medelyan, O., Medelyan, O., Witten, I.: Thesaurus based automatic keyphrase indexing. In Witten, I., ed.: Proc. 6th ACM/IEEE-CS Joint Conference on Digital Libraries JCDL '06. (2006) 296–297