VisualRDR: A general framework for creating, maintaining and learning of ripple down rules for Information Extraction

Delip Rao
CSE Dept, IIT Madras, India
delip@cse.iitm.ernet.in

Sachindra Joshi
IBM India Research Lab
jsachind@in.ibm.com

Ganesh Ramakrishnan
IBM India Research Lab
ganramkr@in.ibm.com

Avishkar Misra
CSE Dept, UNSW, Australia
amisra@cse.unsw.edu.au

Sreeram Balakrishnan
IBM India Research Lab
sreevb@us.ibm.com

Ashwin Srinivasan
IBM India Research Lab
ashwin.srinivasan@in.ibm.com

Abstract

The problem facing the industry and the common user today is that of an information glut. Large amounts of useful information, in various forms, are being generated mostly for human consumption. This deluge of information requires us to find ways of gathering information from such unstructured sources with as little manual intervention as possible. This has been made possible by methods of information extraction. In this paper we propose a novel knowledge-based approach to information extraction. We choose Ripple Down Rules (RDR) as the knowledge acquisition framework since they ensure clear separation of knowledge engineering and domain expertise. We demonstrate how the knowledge for information extraction can be accrued in an incremental fashion in the framework of RDR. We also discuss a graphical user interface (GUI) that assists the domain expert in the creation and the maintainance of this knowledge base.

1 Introduction

Information comes in various forms from various sources including, but not limited to, product manuals, FAQs, man-pages, websites, weblogs, email and news articles. The information thus generated is dynamic. Further to complicate matters, the information in most cases is generated for manual viewing where style and language occlude automatic perception of information. At the time of this writing, Google[1] alone indexes more than eight billion pages. Much more information than this exists in offline form. Manually perusing through this sea of unstructured sources for extracting information is an infeasible task. This motivates the need for techniques that automatically extract relevant information and present them in a structured format to enable querying or further processing. Information extraction is the task of identifying and extracting relevant information from an unstructured source. The information extraction (IE) task can be formally defined as follows [20]:

> "Given a set of structured elements E (target schema) and an unstructured source S, extract all instances of E from S."

In general, there are two approaches to building information extraction systems: the *knowledge engineering approach* and the *trainable system approach* [1]. The knowledge engineering approach refers to manual development of rules (or grammars) for IE by "Knowledge Engineers". The advantage of such a system is that, with skill and experience, it is possible to achieve good performance. Also the resulting knowledge base is very compact and comprehensible. But the flip side is that the manual rule development process, although easy, is very laborious.

The strengths and weaknesses of the automatic training approach are contrary to those of the knowledge engineering approach. The focus here is on pro-

International Conference on Management of Data
COMAD 2005b, Hyderabad, India, December 20–22, 2005

[1] `http://www.google.com`

ducing training data instead of producing rules. This training data is used to learn a model that could be used to process novel instances. With the help of domain-experts to annotate texts, systems can be customized to a specific domain without any intervention from the developers. The success of this method depends on the availability of good training data. Annotated training data may be sparse, or difficult and expensive to obtain. Despite this, contemporary work focuses on the automatic learning aiming to alleviate the aforementioned problems.

Recent study by Ireson *et al* [13] shows that the adoption of a machine learning algorithm, in itself, does not provide a guaranteed advantage in information extraction. Hence we aim to bring the best of both worlds together by combining the trainable system approach and the knowledge engineering approach. We propose a framework for acquiring rules for information extraction which can later be refined by a knowledge engineer to "tune" the system for high performance. This is based on Compton *et al's* seminal work [4] on a knowledge acquisition methodology called the Ripple Down Rules (RDR). Our inspiration to use ripple down rules comes from its rich structure due to exceptions. Exceptions allow addition of knowledge without breaking the previous rules.

Section 2 describes the prior art in Information Extraction systems. Section 3 introduces the knowledge acquisition framework for information extraction. Although the focus of this paper is on using ripple down rules for information extraction, in Section 4, we also present a method of automatically learning such rules from a raw corpus. The knowledge acquisition framework was implemented as a visual tool, called VisualRDR. Section 5 describes VisualRDR. The experiments and the results of using RDR for information extraction are detailed in Section 6.

2 Information Extraction

In this section we provide a synoptic view of some of the existing work on information extraction. Information extraction approaches for the purpose of this study are roughly classified into pattern based approaches and statistical approaches.

2.1 Pattern based approaches

The basic idea behind pattern based approaches is to learn patterns that can extract the relevant information. The information extraction process is treated as a process of slot-filling. The patterns (or rules) can be used either for single slot extraction or multi slot extraction. In multi slot extraction the extraction rules are able to link together related information, as opposed to single slot rules that can only extract isolated data (e.g. in a document that contains several names and addresses, single-slot rules can not specify what the address of a particular person is). Section 3 lists some examples of such systems.

2.2 Statistical approaches

Machine learning methods using Naive Bayes, Support Vector Machines (SVM), Hidden Markov Models (HMM), Conditional Random Fields (CRF) and Semi Markov CRFs have been applied to information extraction. In the Naive Bayes method [7] we treat the document as a bag of words and totally ignore the linguistic structure of the document and a priori probabilities are estimated using weights taken as the TF-IDF measure of the words. Experiments have been done to study the possibility of using support vector machines in information integration. Features from the document/sentence are extracted and classified using an SVM classifier. Hidden Markov Models are a powerful probabilistic tool for time series data and have been successfully applied to sequence labeling and prediction problems in speech and text processing. Recently, McCallum *et al* [12] have used HMMs to extract information from natural language text. They alleviate the large training data requirement in HMMs using a statistical technique called shrinkage. Peng *et al* [14] have used conditional random fields to extract information from scientific papers and Sarawagi *et al* [21] demonstrate the possibility of using semi Markov conditional random fields.

3 Knowledge acquisition for Information Extraction

Knowledge acquisition is "the transfer and transformation of potential problem-solving expertise from some knowledge source to a program." [2] Knowledge acquisition is performed by knowledge engineering techniques. Knowledge engineering is "the process of reducing a large body of knowledge to a precise set of facts and rules." [6]

The information extraction task can be solved using a knowledge based approach. Explicit rules could be elucidated for extracting information from unstructured sources. Several efforts in the past have made use of rules for information extraction. Some examples include, AUTOSLOG [18], LIEP [10], CRYSTAL [24], WHISK [23], RAPIER [3] and SRV [8]. All these systems suffer from the maintainability problems pointed by [5].

3.1 Organizing knowledge

Having selected a knowledge based approach to information extraction, we had several options to organize this knowledge. Some options that we considered include decision lists, decision trees, case based reasoning (CBR) and ripple down rules(RDR). Decision lists [19] maintain rules as a series of **if-then-else** statements as shown in Figure 1. Although this method is ap-

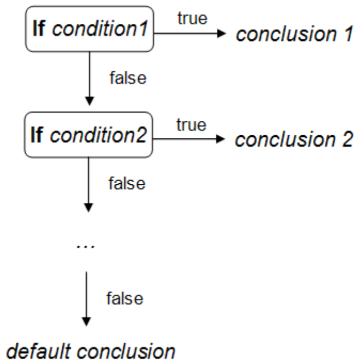


Figure 1: **Decision Lists**

pealingly simple, it suffers from a major setback that rules (**if-then** blocks) cannot be added freely into the knowledge base without risking its integrity. An addition, modification or deletion of one rule could potentially affect others. Hence such knowledge bases would become brittle with use. Another possibility was to use decision trees [15]. Decision trees require well defined attributes and their values and they test a subset of attributes before arriving at a decision as shown in Figure 2. The problem with decision trees is that that

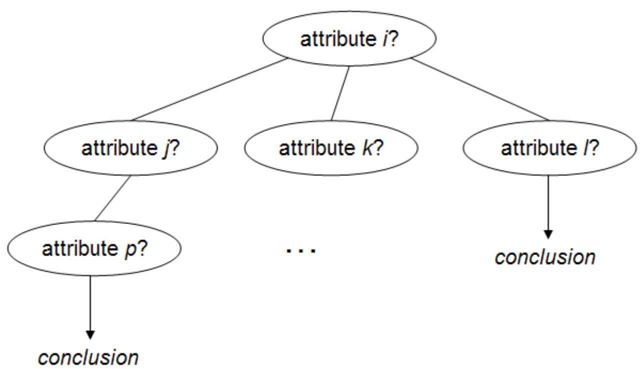


Figure 2: **Decision Trees**

dont allow easy modification. We wanted to build a system that could be used by a domain-expert who may not be even familiar with knoweledge engineering techniques. Case based reasoning (CBR) [11] has been extensively used capturing knowledge. Although it is very domain-expert friendly, it requires considerable investment in the design of case features and similarity metric that enable querying and retrieval of solutions. We use the knowledge acquisition methodology proposed by Compton *et al* [4] called Ripple Down Rules (RDR). The main advantages of using RDR is that

1. RDR enable incremental acquisition of rules.

2. Rules are acquired in context

3. Rule based systems modeled on RDR are amenable to easy maintainance.

4. They have a compact representation than decision trees or case bases.

3.2 Ripple Down Rules

Ripple down rules (RDR) is a knowledge acquisition methodology and a way of structuring knowledge bases which grew out of long term experience of maintaining an expert system , GARVIN-ESI by Compton *et al* [4]. In the RDR framework, the human expert's knowledge is acquired based on the current context and is added incrementally. Ripple Down Rules consist of rules which form a tree structure. Many rules in RDR are exceptions to other rules. Ripple down rules methodology allows incremental changes to the knowledge base without causing unwanted side effects to the existing knowledge base. Compton *et al* have shown [5] that this approach allows clear separation of knowledge engineering and domain expertise. This enables domain experts to change the knowledge base without the need of a knowledge engineer. The root of the tree provides a default conclusion if the rule linked to the child is not satisfied for a particular case. Whenever the RDR incorrectly classifies a case or fails to classify a case, a rule is added. When a rule is added to the RDR structure, the case that prompted the rule is also stored in association with the rule. These cases are called "cornerstone cases." Apart from providing contextual information, later we shall see that the cornerstone cases are also useful in *on the fly* validation. Inferencing in RDR is very simple. Whenever a case is satisfied by the rule which does not have a dependent, *i.e.*, no branches, the conclusion associated with that rule is asserted. On the other hand, if the rule has a TRUE branch then that branch condition is also tested in a depth-first manner. The conclusion of the deepest satisfying node is returned as the result. If besides the default rule, no rule satisfies the case, an exception branch is added to the default rule and a new rule is created. As always, with every new rule we also store the cornerstone case that actuated the creation of that new rule.

3.3 RDR for information extraction

Simple regular expression patterns based on features in the text are used for extracting information. We organize the rules in a ripple down representation with the more generic rules at the top, and having child rules that further specialize them. We use XML to represent the rules for information extraction. The schema for our representation is shown below.

```
<!DOCTYPE KNOWLEDGE [
 <!ELEMENT KNOWLEDGE (RULE+)>
 <!ELEMENT RULE (CONTEXT, CONCLUSION,
                 CSCASE+, EXCEPTION)>
 <!ELEMENT CONTEXT (#PCDATA)>
 <!ELEMENT CONCLUSION (#PCDATA)>
```

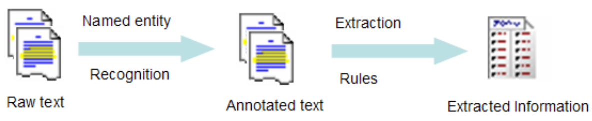


Figure 3: **Information Extraction Process**

```
 <!ELEMENT CSCASE (LOCATION)>
 <!ELEMENT EXCEPTION (RULE+)>
 <!ELEMENT LOCATION (#PCDATA)>

 <!ATTLIST RULE ID CDATA #REQUIRED>
 <!ATTLIST CSCASE ID CDATA #REQUIRED>
]>
```

3.4 On the fly validation

It was realized that the error rate could be eliminated by validating the rules as they were added [4]. Every time a new rule is added, the cornerstone cases could be used to verify if the new rule did not inadvertently break previously existing rules. Thus each time a new rule is added, we verify if the cornerstone cases still continue to be satisfied by the rules which were originally created to cater to them. Instances where this does not happen are flagged to the user as potential problems. The user can then choose to modify the newly added rule or refine the existing rules. This ensures that the knowledge base is consistent at all times.

3.5 Our approach to information extraction

We identify and distinctly separate the tasks of low-level entity tagging and information extraction. By low-level entity tagging, we refer to the task of identifying the category of single or compound words in the raw text solely based on the orthographic, part of speech and dictionary based features of the words. The raw corpus is run through a rule based, custom developed named-entity (NE) tagger [16, 17]. The NE annotator identifies entity categories such as `NAME, PERSON, DATE, AMOUNT, ORGANIZATION` etc.

We then write regular expression patterns with associated actions, over the annotated document. For example, the pattern `ORGANIZATION.*sold.*ORGANIZATION` with an associated action `<seller>ORGANIZATION</seller>.* sold.*<acquired>ORGANIZATION</acquired>` would imply that the organization on the left sold the organization on the right to some third company that happened to be the buying/acquiring company. A collection of such rules organized using the RDR framework is then applied for annotating documents. This process is outlined in Figure 3.

4 Learning of RDR

It is clear that RDR provides an efficient way of organizing rules. Creation of these rules, however, could be a time-consuming task, especially when we already have a huge collection of labeled documents and we need to write rules that have sufficient coverage of labeled documents. We glean these rules from the corpus and construct an RDR tree using the rules. Several efforts in the past, including, INDUCT [9], Cut95 [22] have addressed the problem of learning RDR from training data.

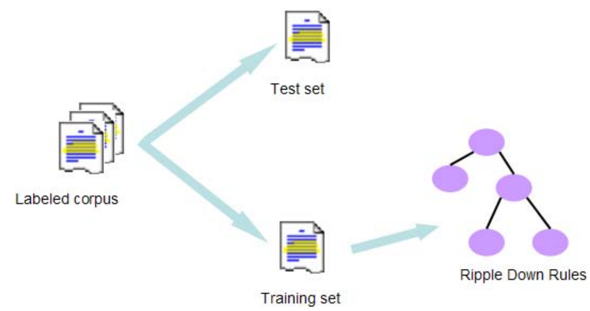


Figure 4: **Learning of Ripple Down Rules**

Figure 4 outlines our methodology. In order to learn we start with a labeled corpus. We split the labeled corpus into a test and training data set. RDR is induced from the training data set and test set is used for validation of the learnt RDR. The RDR is said to cover a case if it accepts the case.

The steps involved in learning the RDR are as follows:

1. Named Entity Recognition is performed on the corpus.

2. Features between entities of interest are identified.

3. These features are clustered based on the criterion such that similar feature patterns fall in the same cluster.

4. A regular expression is formed for each cluster and a first level RDR is built using the regular expression corresponding to each cluster.

5. The first level RDR is applied on the training data.

6. At each node where mis-classifications take place, all the misclassified instances at that node are collected and steps 2-5 are performed on those instances.

7. Step 6 is repeated till no appreciable improvement in the performance of the RDR is observed.

5 VisualRDR

We implemented a graphical user interface (GUI) based tool called VisualRDR that uses RDR for information extraction and provides for easy creation, maintenance and learning of Ripple Down Rules. To enable the tool to be general enough to support any application that uses RDR, a plug-in based architecture is designed so that the actual task to be performed by the rules is factored out into externally loadable plug-ins. Users of the tool can extend it to applications other than information extraction by writing appropriate plug-ins. Further, we provide facilities to learn RDRs if the application wishes to do so. Since ripple down rules are highly interactive, we envision knowledge engineers to manually add, modify and delete rules. Any modifications to the rule-base is reflected immediately in the GUI and the tool is capable of pointing out if any rules broke-down due to the changes made. The following features were identified to be core for any RDR based application.

1. Create a new RDR
2. Edit an RDR
3. Use an RDR against a set of cases
4. On the fly validation

Optionally, we could also add capability to automatically learn the rules. Again, the exact method of rule learning is left to the user.

5.1 Creating new rules

New rules are added to the ripple down rule structure as exceptions. Every new rule added has to be an exception to some previously existing rule, which the newly added rule specializes. In case there is no rule relevant to the current case, the new rule is added as an exception to the default rule, which is the root of the ripple down tree. Figure 5 shows the VisualRDR

Figure 5: **Adding exception**

interface for adding rules.

5.2 Editing an RDR

VisualRDR supports manual editing of ripple down rules, since any knowledge based system will invariably involve a human in the loop. The interfaces are kept simple so that any domain-expert can easily modify, add or delete the rules without the need of a knowledge engineer. We ensure that the consistency of the knowledge base is not perturbed during each edit operation, by providing appropriate visual cues, using *on the fly* validation.

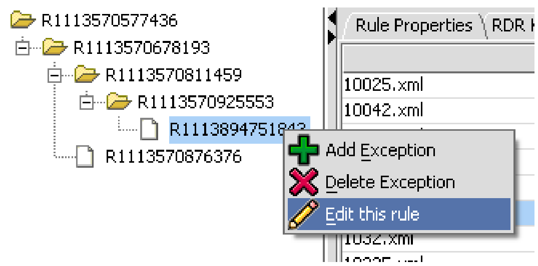


Figure 6: **Editing of rules in VisualRDR**

5.3 Using the RDR

VisualRDR operates in two modes, *batch* and *single*. In the *batch mode*, an entire collection of cases could be classified using the RDR. On the other hand, the *single mode* enables the validation of a single case using the RDR. Figure 7 shows the VisualRDR interface for

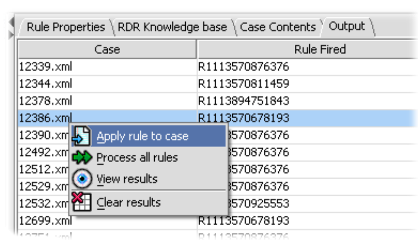


Figure 7: **Using RDR**

applying a rule to a case. The exact operation of the rule is left to the writer of the plug-in. In our case, we wrote a plug-in that enables extraction of information.

5.4 On the fly validation

Each time a change is made to the ripple down rule structure, the rule-base is validated as mentioned in Section 3.4. $OTFValidate$, the procedure for on-the-fly validation is given below. Each time a rule is added to the RDR knowledge base, all cornerstone cases are reclassified using the new RDR. Cases for which the new classification is different from the previous classification are flagged as possible errors.

procedure $OTFValidate(rdr)$

$cstones \leftarrow rdr.cscases$

```
for all case ∈ cstones do
    newrule ← Classify(case, rdr)
    if newrule ≠ case.rule then
        Flag case as possibly misclassified.
    end if
end for
```

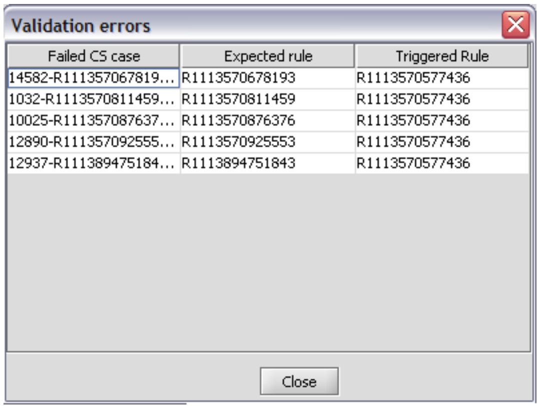


Figure 8: **On the fly validation**

Figure 8 shows on the fly validation being done in VisualRDR.

5.5 Learning

VisualRDR leaves learning of RDR to be implemented by the plug-in writer. In our case, we implement the learning algorithm mentioned in Section 4.

6 Experiments and results

6.1 The dataset

For the purpose of this experiment we use the acquisitions corpus of Reuters news articles on mergers and acquisitions provided by RISE[2]. The corpus has 600 news articles about acquisitions providing information about buyers, sellers, the money involved and so on. The corpus is manually annotated so that it can be used for training and validation purposes.

6.2 The task

Our aim was to identify the names of the seller and purchaser involved from every news snippet in the acquisitions corpus. Although we only extract seller and purchaser names instead of all information extracted by comparing techinques [8] [12], extracting other information is merely a process of adding more rules to the knowledge base. In this work we choose to demonstrate the concept by extracting seller and purchaser information.

[2] `http://www.isi.edu/info-agents/RISE/`

6.3 The Knowledge Base

We performed our experiments with 6 seed rules that were refined over a period of time. These rules were based on simple regular expressions on data annotated by entity tagging. All rules were hand-crafted. For details, please refer to the appendix A.

6.4 Results

The RDR mentioned listed in appendix A was used to extract information from the acquisitions corpus after entity identification. The performance for extracting seller and purchaser information was as shown below. The number of cases in which the seller or buyer information was provided in the labeled collection is shown beside "Total cases". By precision we mean the proportion of the extracted information (seller or buyer) that is relevant and recall is the proportion of the relevant information that was extracted. The F1 measure is a harmonic mean of precision and recall.

	Seller	**Purchaser**
Total cases	471	447
Precision	64.54	58.38
Recall	77.70	80.98
F1 measure	70.51	67.84

Table 1: Results with RDR listed in appendix A

7 Conclusions and future work

In our current work we have shown the utility of a knowledge acquisition methodology, called Ripple Down Rules, for information extraction. We also described VisualRDR, a framework for creating, maintaining and learning of Ripple Down Rules. These rules could be fine-tuned by a domain-expert for achieving high levels of performance. We wish to add additional features to VisualRDR such as highlighting redundant and contradictory rules.

8 Acknowledgements

Delip Rao would like to acknowledge the support by IBM India research lab from May 2005 to August 2005 during which this work was done.

References

[1] D. E. Appelt and D. J. Israel. Introduction to information extraction technology. *IJCAI*, 1999.

[2] D. B. Buchanan, J. B. R. Betchel, and W. Clancey. Constructing an expert system. building expert systems, 1983.

[3] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *AAAI/IAAI*, pages 328–334, 1999.

[4] P. Compton and R. Jansen. Knowledge in context: a strategy for expert system maintenance. In *Proceedings of the 2nd Australian Joint Artificial Intelligence conference*, volume 406 of *Lecture Notes in Artificial Intelligence*, pages 292–306, Adelaide, 1988. Springer.

[5] P. Compton and R. Jansen. A philosophical basis for knowledge acquisition, Aug. 16 2000.

[6] E. A. Feigenbaum. Knowledge engineering. the applied side of artificial intelligence, 1984.

[7] D. Freitag. Using grammatical inference to improve precision in information extraction. In *Workshop on Grammatical Inference, Automata Induction, and Language Acquisition (ICML'97), Nashville, TN*, 1997.

[8] D. Freitag. Multistrategy learning for information extraction, 1998.

[9] B. R. Gaines and P. Compton. Induction of ripple-down rules applied to modeling large databases. *J. Intell. Inf. Syst*, 5(3):211–228, 1995.

[10] S. B. Huffman. Learning information extraction patterns from examples. In *Learning for Natural Language Processing*, pages 246–260, 1995.

[11] D. B. Leake, editor. *Case-Based Reasoning*. The MIT Press, Cambridge, Massachusetts, 1996.

[12] A. McCallum and D. Freitag. Information extraction with HMMs and shrinkage. In *AAAI'99 Workshop on Machine Learning for Information Extraction*, 1999.

[13] F. C. Neil Ireson, D. F. Mary Elaine Califf, N. Kushmerick, and A. Lavelli. Evaluating machine learning for information extraction. In *International Conference on Machine Learning(ICML)*, 2005.

[14] F. Peng and A. McCallum. Accurate information extraction from research papers using conditional random fields. In *HLT-NAACL*, pages 329–336, 2004.

[15] R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[16] G. Ramakrishnan. *Word Associations in Text Mining*. PhD thesis, IIT Bombay, 2005.

[17] G. Ramakrishnan, S. Joshi, S. Negi, R. Krishnapuram, and S. Balakrishnan. Automatic sales lead generation from web data. In *The 22nd International Conference on Data Engineering, Atlanta, GA, U.S.A*, 2006.

[18] E. Riloff. Automatically constructing a dictionary for information extraction tasks. In *AAAI*, pages 811–816, 1993.

[19] R. Rivest. Learning decision lists. *Machine Learning*, 1987.

[20] S. Sarawagi. Automation in information extraction and integration. *VLDB*, 2002.

[21] S. Sarawagi and W. W. Cohen. Semi-markov conditional random fields for, June 07 2004.

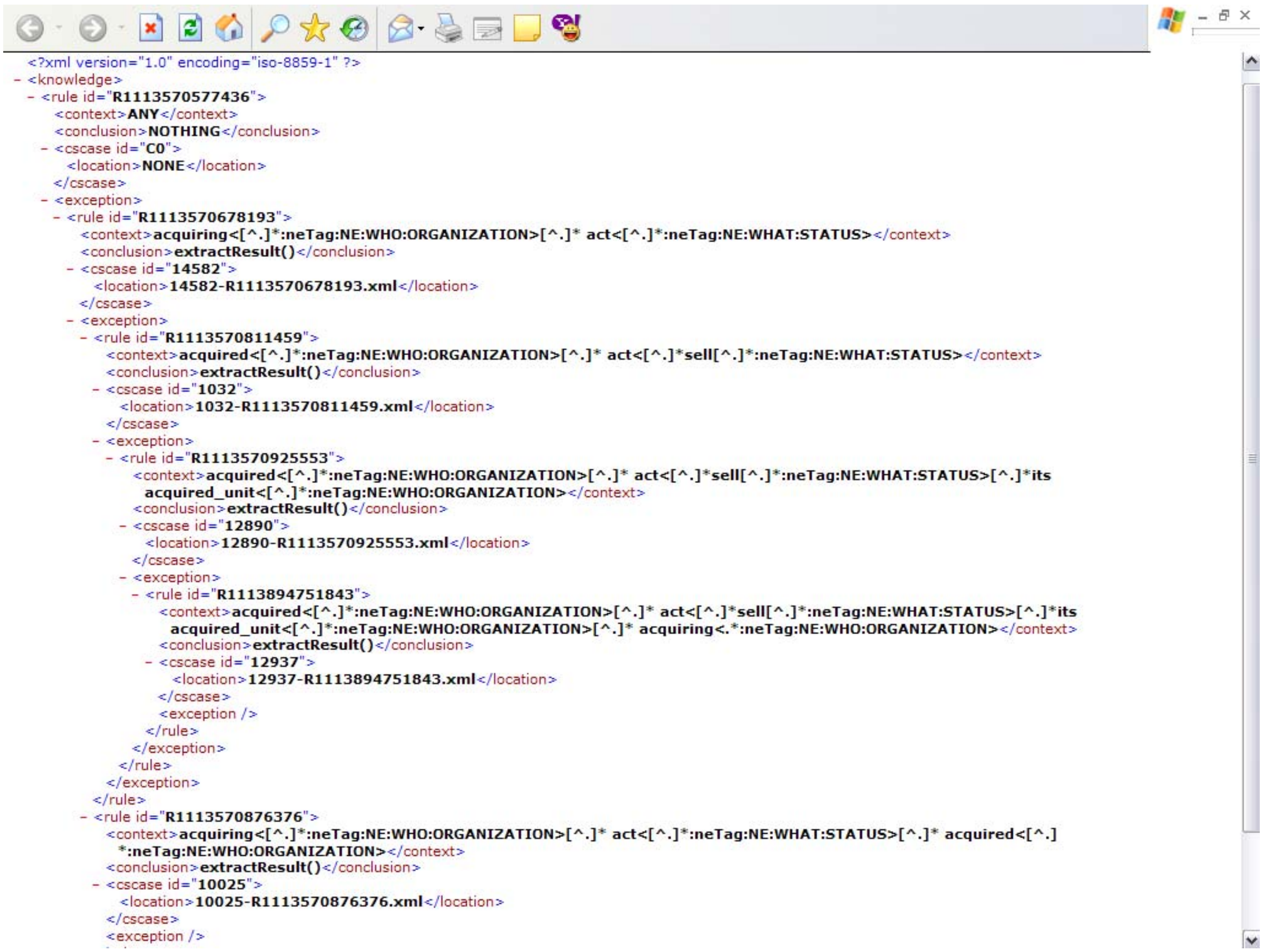
[22] T. Scheffer. Algebraic foundation and improved methods of induction of ripple down rules, Dec. 05 1996.

[23] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34:233, 1999.

[24] S. Soderland, D. Fisher, J. Aseltine, and Lenhert. CRYSTAL: Inducing a conceptual dictionary. *Proc. Int. Joint Conf. Artcicial Inteligence (IJCAI-95)*, pages 1314–1319, 1995.

A The knowledge base

Figure 9 shows a snapshot of the RDR knowledge base used for extraction.



```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <knowledge>
  - <rule id="R1113570577436">
      <context>ANY</context>
      <conclusion>NOTHING</conclusion>
    - <cscase id="C0">
        <location>NONE</location>
      </cscase>
    - <exception>
      - <rule id="R1113570678193">
          <context>acquiring<[^.]*:neTag:NE:WHO:ORGANIZATION>[^.]* act<[^.]*:neTag:NE:WHAT:STATUS></context>
          <conclusion>extractResult()</conclusion>
        - <cscase id="14582">
            <location>14582-R1113570678193.xml</location>
          </cscase>
        - <exception>
          - <rule id="R1113570811459">
              <context>acquired<[^.]*:neTag:NE:WHO:ORGANIZATION>[^.]* act<[^.]*sell[^.]*:neTag:NE:WHAT:STATUS></context>
              <conclusion>extractResult()</conclusion>
            - <cscase id="1032">
                <location>1032-R1113570811459.xml</location>
              </cscase>
            - <exception>
              - <rule id="R1113570925553">
                  <context>acquired<[^.]*:neTag:NE:WHO:ORGANIZATION>[^.]* act<[^.]*sell[^.]*:neTag:NE:WHAT:STATUS>[^.]*its
                    acquired_unit<[^.]*:neTag:NE:WHO:ORGANIZATION></context>
                  <conclusion>extractResult()</conclusion>
                - <cscase id="12890">
                    <location>12890-R1113570925553.xml</location>
                  </cscase>
                - <exception>
                  - <rule id="R1113894751843">
                      <context>acquired<[^.]*:neTag:NE:WHO:ORGANIZATION>[^.]* act<[^.]*sell[^.]*:neTag:NE:WHAT:STATUS>[^.]*its
                        acquired_unit<[^.]*:neTag:NE:WHO:ORGANIZATION>[^.]* acquiring<.*:neTag:NE:WHO:ORGANIZATION></context>
                      <conclusion>extractResult()</conclusion>
                    - <cscase id="12937">
                        <location>12937-R1113894751843.xml</location>
                      </cscase>
                      <exception />
                    </rule>
                  </exception>
                </rule>
              </exception>
            </rule>
          - <rule id="R1113570876376">
              <context>acquiring<[^.]*:neTag:NE:WHO:ORGANIZATION>[^.]* act<[^.]*:neTag:NE:WHAT:STATUS>[^.]* acquired<[^.]
                *:neTag:NE:WHO:ORGANIZATION></context>
              <conclusion>extractResult()</conclusion>
            - <cscase id="10025">
                <location>10025-R1113570876376.xml</location>
              </cscase>
              <exception />
```


Figure 9: RDR knowledge base used for information extraction