Nodose Version 2.0

Brad Adelberg
Computer Science Department
Northwestern University
adelberg@cs.nwu.edu

Matthew Denny
Computer Science Department
Northwestern University
mattd77@nwu.edu

ABSTRACT

This paper describes a tool, called Nodose, we have developed to expedite the creation of robust wrappers. Nodose allows non-programmers to build components that can convert data from the source format to XML or another generic format. Further, the generated code performs a set of statistical checks at runtime that attempt to find extraction errors before they are propogated back to users.

Keywords

Wrapper generation, data extraction, error detection.

1. INTRODUCTION

This paper describes Nodose, a GUI-based tool for building robust translators, extractors, and wrappers for text sources. By robust we mean that the generated code is able to detect extraction errors even once it's deployed so that users of the extracted data can be confident in its quality. Even assuming bug free code, there are three sources of errors that a deployed wrapper might encounter:

1. The format of the wrapped source may change. Web sites are particularly guilty of this.
2. Although a wrapper tests successfully on a set of example data, it is possible that it will fail in actual operation if it encounters a document with a previously unencountered feature, such as an infrequent field. Given that wrappers often need to be developed very quickly, particularly after a previously supported web site has changed formats, the likelihood of not testing with a fully representative set of pages increases.
3. For sources where the documents are human generated or OCRed, typos or OCR errors can wreak havoc with the wrapper's parser.

It is critical that wrappers that are part of production systems be robust in the face of the problems above. Performance should degrade gracefully in the presence of errors, and even more importantly, errors must be detected before corrupt data is returned to the user.

2. EXAMPLE

We will describe how Nodose can be used to extract data from a web site using the Internet Movie Database as an example. The extraction process begins by loading a single page into Nodose. The user then hierarchically decomposes the structure within the page using a GUI and an automatic parsing component. Next, additional pages corresponding to other movies are loaded into the system and automatically parsed. Any errors are corrected by using the GUI. The process is complete when all of the pages have been successfully parsed. Once the rules for parsing the web pages have been inferred, the extracted data can be output in many different forms. Options in the current implementation include XML and a tab-delimited format that can be imported into spreadsheets and relational database systems for tabular sections of the extracted data. Also, a standalone component can be automatically generated that parses documents and outputs XML or any of the other reporting formats.

The first step in decomposing a movie page is indicating its top-level structure. We can view a single page as a record with many fields. Some of the more interesting fields are shown already extracted in Figure 1. The user will enter each of the fields into Nodose by selecting the relevant portion of the text in the document window and clicking on the "add structure" button in the tool bar (see Figure 1). The type, type name, and label of each field can be entered using the controls on the bottom portion of the window. Since some of the fields of the record are complex types (lists), the decomposition process must continue.

Suppose the user chooses to decompose the list of cast members next. Double clicking on that node in the tree view panel will display only the portion of the document mapped to the results list. The user then selects the text of the first element of the list and adds this as a structure. Next, the second element of the list is added. The user could continue to add every element in this manner but this would become tedious if there were many elements. Instead, the user can ask Nodose to try to infer the remaining elements by mining the text. If the tool mistakenly identifies elements, the user can correct a few of the errors and ask that the text be remined. In this way, the correct grammar for the component will eventually be learned. Once it is, Nodose is able to identify all of the other elements of the list correctly.

The decomposition process would continue until every interesting structure in the web page had been identified. At that point, Nodose is loaded with additional pages. These are automatically parsed using the grammar inferred from the first page. It's possible, though, that parsing will fail on one of the additional. For example, suppose that the second movie is in

multiple languages. Nodose may misparse the languages list, in which case the user must correct the parsed tree of the new file, describing the list using the GUI as before. The extractor will then update its grammar to account for the multiple element list so that future pages will be parsed correctly.

In order to help find direct users to extraction errors, a statistical analyzer (described in Section 3) is run after the miner. Nodes that are flagged as questionable are shown in red; those that are not questionable are shown in green. Also, the analyzer results box will contain details of the check that failed for flagged nodes. An example is shown in Figure 1 in which the cast list has been flagged since its range is more than 1 standard deviation longer than other cast lists (from other pages).

The user should examine nodes shown in red to determine if they were mined incorrectly. If so, the node should be edited by hand and then remined. If the node is correct but is marked in red, the user checks the "node is correct" box which notifies the analyzer that it must loosen its thresholds. After a number of example pages have been correctly parsed and their nodes correctly analyzed, the analyzer will be loose enough to avoid most false positives but strict enough to catch most true errors.

3. DETECTING ERRORS

The error reporting system we have implemented is based on finding statistical outliers, or instances of a type that have characteristics that deviate greatly from the average instance of that type. For instance, if 50 instances of a given type are 10 characters long and a new instance is found with 200 characters, the extractor has probably made a mistake.

In order for the system to find statistical outliers, the following statistics are incrementally maintained per type:

- *average node range* - The length of the node.
- *percentage of a node covered by its children* (record and collection types only) - Average percent of node range that is covered by its children.
- *percentage of records with field* (record types only) - The percentage of nodes of a given record type in which a certain field is present. For example, 90% of the employee records have a salary field.
- *average number of unused fields* (record types only) - The average number of fields used in a record type subtracted from the number of fields defined for the record. If records always contain every field, this number will be 0.
- *percentage of time field A precedes field B* (record types only) - The percentage of records of a given type in which field A directly precedes field B. For example, name might precede birthdate in 80% of all employee records.
- *average number of children* (collection types only) - The average number of elements in a collection.
- *list of acceptable characters* (atomic types only) - For atomic nodes, Nodose stores a list of all of the permissible characters. The default starting lists for all of the atomic types are shown in Table 1.

The statistics are generally kept as a series of running totals. This implementation not only bounds the space used by the type information, it also keeps the statistical information independent from the text of the documents, thus allowing the statistical data to persist even in the presence of impermanent documents (e.g. outdated documents that are deleted, documents entering the system as streams but never materialized as a file, etc.).

Atomic Type	Initial Legal Characters
Integer	'0'-'9'
Float	'0'-'9','.','E','+','-'
String	'a'-'z','A'-'Z'
Date	'0'-'9','/','-'
Email address	'a'-'z','A'-'Z','0'-'0','.','@'
URL	'a'-'z','A'-'Z','0'-'0','.','~'

Table 1: Initial lists of legal characters by atomic type.

Whenever new nodes are parsed, the analyzer checks the newly created nodes to see if any is a statistical outlier. If any of the node's attributes lie outside of the threshold, one or more warning messages are generated for that node and the node is marked in red in the GUI. From the user standpoint, a flagged node in the GUI means that the node is an error or a false positive generated by the analyzer. If the flagged node is indeed an error, the user should edit the node manually, and mine the node again so that the miner may learn the new variant on the structure. If the flagged node is not an error, it is a false positive; a node that is not an error, but still a statistical outlier. To correct this problem, the user should explicitly accept the node (by clicking on a checkbox). If the node is accepted, the analyzer thresholds will be expanded to accommodate the node. For example, consider a node of type A that has a range that lies 1.5 standard deviations away from the mean. If the node is accepted and the current threshold for the range of that type is 1.0 standard deviations, then the threshold is expanded to 1.5 standard deviations.

The threshold information maintained by the analyzer can be used by stand-alone wrappers. When the wrapper generated by Nodose receives warning messages from the analyzer, they are written out to stable storage so that users may check for potential problems in documents mined non-interactively. They can also be included as attributes if the data is written out as XML.

4. RELATED WORK

We are not aware of any other work that explicitly addresses the correctness of extracted data. The most similar work in spirit is that of [6] in which the authors attempt to learn rules for data mining that are robust in the face of changes to the underlying data. The authors propose a metric for the robustness of a rule and use the metric to prune the rules produced by the data mining algorithms.

While there is very little work on the correctness of extracted data, there has recently been a flurry of work on extraction systems in general [2,3,4,5,7]. Due to space constraints (and given that the focus of this paper is not Nodose in general), we cannot address these systems here, and instead refer the interested reader to the cited papers.

5. REFERENCES

[1] B. Adelberg. "NoDoSE - A Tool for Semi-Automatically Extracted Structured and Semistructured Data from Text Documents." In *Proceedings of SIGMOD*, Seattle, WA. 1998.

[2] N. Ashish and C.A. Knoblock. "Wrapper generation for semi-structured Internet sources." In *Workshop on Management of Semistructured Data*, 1997.

[3] J.-R. Gruser, L. Raschid, M. E. Vidal, L. Bright. "Wrapper Generation for Web Accessible Data Sources." In *Proceedings CoopIS*, 1998.

[4] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. "Extracting semistructured information from the web." In *Workshop on Management of Semistructured Data*, 1997.

[5] C. Hsu. "Initial Results on Wrapping Semistructured Web Pages with Finite State Transducers and Contextual Rules." In *Proceedings of AAAI'98 Workshop - AI and Information Integration*, 1998.

[6] C. Hsu and C.A. Knoblock. "Discovering Robust Knowledge from Databases that Change." Available at www.isi.edu/sims/chunnan.

[7] N. Kushmerick, D.S. Weld, and R. Doorenbos. "Wrapper induction for information extraction." In *Proceedings of IJCAI*, 1997.

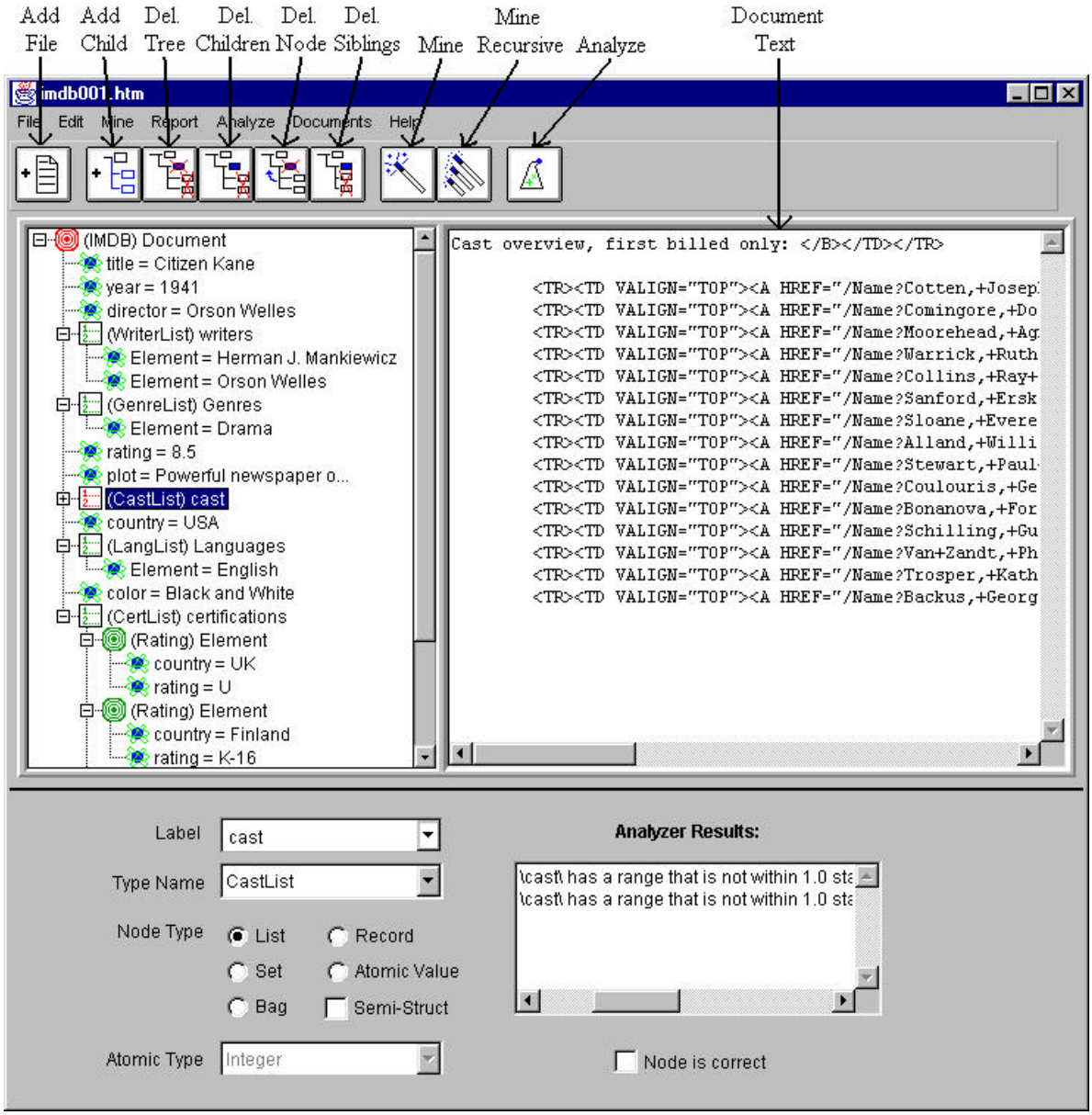

Figure 1: Screenshot of Nodose wrapping the Internet Movie Database.