Indexing Real-World Data using Semi-Structured Documents

Albrecht Schmidt, Menzo Windhouwer, Martin Kersten

Information Systems (INS)

INS-R9902 March 1999

Report INS-R9902
ISSN 1386-3681

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Indexing Real-World Data using Semi-Structured Documents

Albrecht Schmidt, Menzo Windhouwer, Martin Kersten
{albrecht,windhouw,mk}@cwi.nl

CWI, P.O.Box 94079, 1090 GB Amsterdam, The Netherlands

ABSTRACT

We address the problem of deriving meaningful semantic index information for a multi-media database using a semi-structured document model. We show how our framework, called *feature grammars*, can be used to (1) exploit third-party interpretation modules for real-world unstructured components, and (2) use context-free grammars to convert such poorly or unstructured input to semi-structured output. The basic idea is to enrich context-free grammars with special symbols called detectors, which provide for the necessary structure *just-in-time* to satisfy a parser look-ahead. A prototype implementation has been constructed in the Acoi project to demonstrate the feasibility of this approach for indexing both images and audio documents.

1991 Computing Reviews Classification System: [H2.4,H.3.1] Multimedia indexing
Keywords and Phrases: multimedia databases, web databases, meta data, semantic indexing, feature detection, content-based retrieval
Note: The Acoi project is funded through the Dutch SION project "Amis" and Telematics Institutes project "Digital Media Warehouses".

1. Introduction

Semi-structured data and their storage in databases have been a research topic for quite a while. This acknowledges the fact that there are an increasing number of documents with characteristics [7] different to those dealt with in traditional database management systems. In this arena, XML [5] is a de-facto standard to bridge the gap between the structured and semi-structured world by elicitation of their semantic structure.

From a theoretical perspective, a variety of formal tools are directly applicable, e.g. Bracketed Context Free Grammars [8] and Regular Right Part Grammars [13]. Combined with their parsing technology, they provide a sound basis to analyse XML documents before being stored in the database, to derive a suitable database scheme, and conversely, to generate semi-structured documents from the structured world managed in a conventional database system.

However, XML documents describe only part of the real-world data. Their innermost components still rely on external components to access and interpret the information. Although in most cases this boils down to (semantic) interpretation of text, it may also call for accessing images, videos, and audio objects managed elsewhere. The predominant approach is to use a sophisticated wrapper/mediator architecture, e.g. [4, 2, 14]. For an overview of conventional approaches, see [7].

Since much of the semantic meaningful information is hidden behind these external structures, content-based indexing has become a major problem. In most practical cases it is impossible to dynamically inspect an object, while it is often not clear what index information can be extracted cheaply.

The novelty of our approach is to tackle the interpretation of such real-world objects as an enriched parsing problem. For this purpose, we introduce the notion of a *feature grammar* that combines the descriptive power of context-free grammars and the flexibility of plug-in detectors. Detectors are functions provided by third parties which map raw data items, e. g. images, to semi-structured data

```
HTTP/1.1 200 OK
Date: Fri, 26 Feb 1999 14:35:52 GMT
Last-Modified: Fri, 26 Feb 1999 14:38:41 GMT
Content-Length: 112
Content-Type: text/html
<html>
 <head>
  <title>
   Acoi
  </title>
 </head>
 <body>
  <img src=./images/AcoiLogo.gif>
 </body>
</html>
```

Figure 1: An example web object

upon demand. The methods presented serve as a means to transparently describe the relationship between raw data and inferred concepts.

Parsers derived from feature grammars are used to built a large index for multi-media data accessible on the World Wide Web. Our long term goals are to explore techniques for incremental index construction in this volatile environment, dealing with queries over an incomplete index, and to manage a large (evolving) database schema. These experiments take place in the context of our high-performance database management system [3].

The focus of this paper lies on exposing our view on how raw data can be mapped onto hierarchical, i. e. semantically enriched data. First, we present a motivational example to show the benefits of feature grammars. It illustrates that it seamlessly matches traditional parsing techniques implemented in compiler compilers such as Yacc [11]. In Section 3, we give a formalization of feature grammars. It will be shown that they fit seamlessly into the concept of context-free grammars if we interpret detectors as 'active nodes'

2. Motivating Example

The benefits of access structures for the world wide web is beyond question. The prime approach is to build a warehouse with access information [7]. Major players in this game are the well known search engines. In many cases a webrobot traverses the web regularly and stores key features for each HTML page encountered, which serve as an index to locate interesting objects. Although the primary document type is still HTML, recent progress in multi-media retrieval techniques makes construction of multi-media search engines attainable [10].

Our conjecture is that effective access to multi-media objects comes from the combination of textual, semantic, and media-specific features. The former is derived from the textual context of an object, while the latter are derived from the multi-media types, i.e. color distribution and textures. Semantic information is obtained from the textual context when this is encoded in XML. Otherwise, it has to be deduced from a combination of primitive features and from manual interaction by a human classifier.

Figure 1 shows an object taken from the web domain: a sample HTTP skeleton page containing a reference to an image logo. The key features of this object, for both the webrobot and the user of a search engine, have to be modeled explicitly. Some parts come from the semi-structured data tags and other parts from interpretation of the real world data source, i.e. the GIF image.

The key observation underlying our indexing scheme is that all features needed for fast access can be modeled by a collection of hierarchical structures [5]. This collection can be unified into a single

```
%start      web_object;

%atom       str url,content_type,title;

%detector   web_header(url);
%detector   image_type ? content_type="image/gif";
%detector   photo;
%detector   logo;
%detector   page_type ? content_type="text/html";
%detector   web_page;

web_object: url web_header web_body;
web_header: content_type;
web_body  : image_type web_image | page_type web_page;
web_image : photo | logo;
web_page  : title? web_object*;
```

Figure 2: A feature grammar for a web object

hierarchical structure to describe the complete access information, which in turn can be conveniently described with a context free grammar [8, 13].

As the search engine needs key features only, a filter step and optional transformation step is needed. This is achieved by making some of the nodes in our grammar *active*. An active node knows about the internal structure of a data source, either containing real world or semi-structured data, and it knows how to find the interesting information.

In our example we have a single document type, namely a HTTP document. This document consists of two parts (see also figure 1): a header and a body [5]. The header contains meta information and the body contains binary or character data of the object described by the meta data. The internal structure of the body depends on the MIME type of the document, which is found in the header.

Figure 2 gives a context free grammar where the active nodes are tagged as *detector*s. The external data source is filtered by the `web_header` detector and transformed by the `photo`, `logo` and `web_page` detectors. A simple walk-through of the parsing process [1] will clarify the use of the detectors and other (non-)terminals found in the production rules.

The parsing process starts with the insertion of a string labeled `url` into the empty token sequence. The start symbol of the grammar is `web_object` and it specifies that such an object is (partially) described by an `url`. As we provided this `url` in the token sequence, this terminal is found, but the rest of the rule remains to be proven valid.

The next part of the rule is a non-terminal, in this case it is the detector `web_header`. This detector takes as argument the partially constructed parse tree, the non-terminal `url`. The `web_header` detector knows how to retrieve the HTTP document and how to parse the contents of its header and body. It filters out the information of interest, the Content-Type and body, and puts them labeled correctly into the token sequence. The parser now proceeds and finds the correct tokens and will accept the `web_header` production rule.

The next special non-terminal in the grammar is the `image_type` detector. This detector inspects the part of the parse tree already known and stores a boolean value, depending on the success of its condition, in the sequence.

This parsing process continues until there are no non-terminals left to be validated. Figure 3 shows

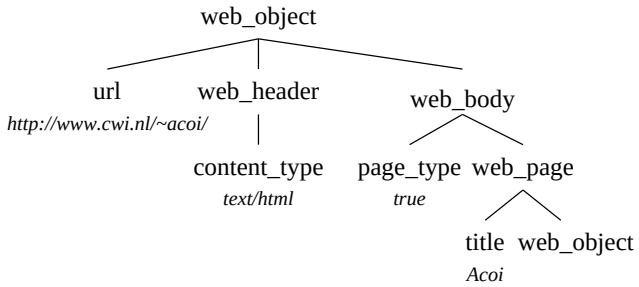

Figure 3: A partial parse tree

the parse tree constructed by this process for the example URL `http://www.cwi.nl/~acoi/`.

As the grammar is recursive this process could go on and, in principle, ultimately result in an index for all multi-media objects on the web.

The process sketched is still strongly linked with conventional parsing methods. A major component of the detectors is to intercept the lexical analysis phase, producing the right tokens for real-world data items just in time. However, this lexical token sequence shows a more dynamic behavior than in the conventional parsing case. For, it can also take into account both the partial parse tree constructed and any relevant information in the environment. The next section provides a formalization of these ideas, and show that the process is still semantically equivalent to conventional parsing methods.

3. Feature Grammars

This section introduces our theoretical framework. First, a formal description of a minimal subset of feature grammars is given. Then we show how this subset can be enriched by introducing regular-expression right-hand sides, multiple views on data items, and the detector concepts.

3.1 Formal description

The goal of this subsection is to show how feature grammars can be made to fit into the schema of context-free grammars. However, there are differences, which force us to change some fundamental definitions to maintain the benefits of formal language theory [9].

There are two main points making the parsing process of feature grammars different to the parsing of 'ordinary' context-free grammars: (1) There is no *a priori* fixed input word. (2) Intermediate symbols may appear and disappear again during parsing and therefore should not be considered part of the language of a feature grammar. (1) is illustrated in the introductory example, (2) can be seen in figure 4(b)–(c).

The basic idea to cope with these problems is to not regard an input $\alpha_1 \dots \alpha_n$ before parsing as part of the language of a grammar, but the transformed input $\beta_1 \dots \beta_n$ after the parsing where each β_i equates with a partial parse tree of the form $\beta_i(\beta_{i,0}, \dots, \beta_{i,k})$.[1] The β_i's are the grammar symbols and replace the input sequence. This allows us to view feature grammars as context-free grammars where lexical values are replaced by their terminal symbol or structure derived by an 'active' nodes.

In the sequel, V denotes a set of variables (non-terminals), T a set of terminals, $S \in V$ the start symbol, P a set of productions, D a set of special variables called detectors. G denotes a grammar, Σ the respective input alphabet, We also use the convention that $\alpha_i \in \Sigma$ and $\beta_i \in T$.

To formalise our ideas, imagine a nondeterministic stack acceptor which parses an input against a feature grammar.

Suppose, we have an initial input $\alpha_1 \dots \alpha_n$ (figure 4(a)) which consists of letters $\alpha_i \in \Sigma$. The part

[1] A linear post-fix representation of the parse tree is used

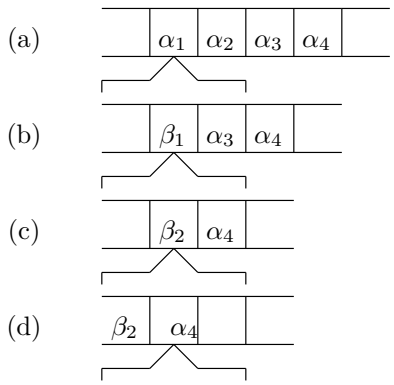

Figure 4: The parsing process

to the right of the read head is denoted by α_i. The part left to the read head has been replaced by a linear representation of the partial parse tree(s) $\beta_{i-1}(\beta_0, \ldots, \beta_{i-2})$ already recognized.

In order to allow efficient parsing, the input must undergo an abstraction procedure, called lexical analysis. Abstractions like '3.1415 is a floating point number' of 'what follows is an image with faces' are done by detectors which translate part of the remaining input stream $\alpha_j \ldots \alpha_k$ into a stream of recognizable terminal symbols $\beta_j \ldots \beta_m$ as in figure 4(b). The resulting stream is consumed by the parser (figure 4(c)), keeping the complete derivation structure.

To formalise the notion of a detector we must take into account that the behavior of a detector is influenced by the complete input stream. Formally, however, it is a mapping of the remainder of an input sequence α_i into a sequence $\beta_{i,0}, \ldots, \beta_{i,l}$.

This leads to the following definition of detector:

Definition 3.1 A *context-free detector* for the non-terminal $d \in V$ is a function $d : \Sigma^* \to T^*$.

Definition 3.2 A *detector* for the non-terminal $d \in V$ is a function $d : (T^*, \Sigma^*) \to T^*$.

The context free detector only looks at the remainder of the input sequence and prepares it for continued parsing. The general detector takes the already recognized portion into account, it produces a context sensitive adjustment.

As the remaining input is altered by (implicit) detectors[2], we need to make changes to the traditional definition of the language $L(G)$ of a grammar. Therefore, we define that the language $L(G)$ of a feature grammar G consists of all $\beta = \beta_1 \ldots \beta_n$ of all possible successful parsing processes.

Note that it is problematic to say that $\alpha_1 \ldots \alpha_n$ is in $L(G)$ as $\alpha_1 \ldots \alpha_n$ is manipulated during the parsing process. To be able to analyse the properties of G we must look at what the read head sees at the time the input is consumed. Actually, this is the word $\beta_1 \ldots \beta_n$.

The advantage of the above definitions is that now the concept of detectors fits seamlessly into the theory of context-free grammars as we will see in the following definitions. To recall, a grammar $G = (V, T, S, P)$ is called *context-free* if all productions in P are of the form $A \to x$, where $A \in V, x \in (V \cup T)^*$. Regarding detectors as 'active' nodes in the derivation tree under construction, we define:

Definition 3.3 We call $G = (V, T, S, P, D)$ a *feature grammar*, if $G' = (V \cup D, T, S, P)$ is a context-free grammar.

Note that the leaves in feature grammar parse trees are not necessarily labelled with terminals but with either terminals or detectors. Clearly, we don't want a detector to output tokens that garble up

[2] we assume built-in detectors for the basic types INT,STR, ... , which model traditional lexical analysis components.

our parsing process; instead, its output should be a valid input to the parser. Therefore we introduce the following nomenclature: We call a detector *safe*, if the token stream after the detector has been called is still parsable. We call a detector *deterministic*, if its output depends only on its input. We call a detector *d restricted*, if during parsing all its output resolves to a partial tree whose root is the detector node.

To be able to talk about partial parse trees, we say that the context-free grammar $G' = (V', T', a, P', D')$ is a minimal sub-grammar of the context-free grammar $G = (V, T, P, S, D)$ if (1) $(V', T', P', D') \subseteq (V, T, P, D)$, (2) $a \in V' \cup T' \cup D'$, and (3) every $v \in V' \cup T' \cup D'$ can be derived from a.

Theorem 3.1 *Let $G = (V, T, S, P, D)$ be a feature grammar with all detectors $d \in D$ safe and restricted. Then the languages of all of G's minimal sub-grammars with $S \in D$ are context-free.*

Sketch of Proof. A first observation is that all rules of the sub-grammar, once they are evaluated, are matched against a fixed token stream. Then there are two cases: (1) no detector is the root of a subtree. (2) a single detector is the root subtree. In case (1) parsing is done as with 'normal' context-free grammars. In case (2) the grammar 'under' the detector node is still context-free and can be parsed. Following this reasoning from the leaves of the parse tree up to the top node proves theorem 1. □

To be able to compare the behavior of feature grammars to that of context-free grammars, we say that a grammar G *behaves* like a context-free grammar if it can be parsed like a context-free grammar.

Theorem 3.2 *A feature grammar G behaves like a context-free grammar.*

Sketch of Proof. Theorem 2 holds because all sub-grammars of G are context-free and because the class of context-free grammars is closed under concatenation and embedding. To see this, let $G_1 = (V_1, T_1, S_1, P_1)$ and $G_2 = (V_2, T_2, S_2, P_2)$ be context-free grammars, and $L(G_1)$ and $L(G_2)$ their languages. Then $L(G) = \{w_1 w_2 | w_i \in L(G_i)\}$ is described by $G = (V_1 \cup V_2, T_1 \cup T_2, S, P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\})$, which obviously is context-free. So $L(G)$ is context-free. The same reasoning holds for embedding a grammar G_1 in a rule of G_2. □

To sum up, we can see that there are many similarities between context-free grammars and feature grammars. See [6] for an efficient way of parsing.

3.2 Multi-Media Indexing

Most of the data produced by computers have some inherent structure which may not even be what is meant by semi-structured. But it is often easy to formulate some basic rules; this is where feature grammars can be applied. Even in the case of data for which no formulation of rules seems feasible, like raster-data, audio etc., detectors can be applied like filters whose output is then integrated into the derivation tree. In the Acoi [12] project, filters for raster and MIDI data have been developed.

We have yet to mention that detectors actually come in two flavours: *black box* and *white box* detectors. Black-box detectors are generally software modules whose implementation is hidden; only the structure of their output is known. In contrast, the functionality of white-box detectors is specified in the grammar with an OQL-like full programming language that has enough flexibility to query already built branches of the parse tree and deduce information from them. Also note that, from a theoretical point of view, all black-box detectors may be substituted by white-box detectors.

There are several additions we have made in our implementation which are convenient to the user. To facilitate the concise formulation of abstract ideas we allow regular-expression like left hand sides in the grammar [13]. Also, we provide facilities for defining multiple indexes over the same data items (see the definition of `web_image` in the example).

The advantage of feature grammars is that they allow us to define all levels of abstraction in a single uniform framework which tightly integrates processes like 'lexical analysis', alternative views, and derivation of higher order concepts. So there are two quite distinct applications of feature grammars. (1) The grammar is a condense description of the search index information needed. (2) The grammar is the semantic framework to articulate meaningful queries over the database.

4. Conclusion

We have presented a practical and flexible method to integrate barely structured data and plug-in modules in a semi-structured context. We have shown that from a theoretical point of view, they can be viewed as context-free grammars with active nodes that make changes to the input sequence.

The current status of the project is that a compiler compiler has been implemented; the system is currently used for indexing web pages, images, and MIDI files.

Future work will include topics like incremental parsing of documents, schema transition and a drive towards XML which can described and enriched conveniently by feature grammars. Also, we are planning to work on strategies to accelerate the parsing process.

References

1. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers : Principles, Techniques, and Tools*. Addison-Wesley, 1985.
2. K. D. Bollacker, S. Lawrence, and C. L. Giles. Citeseer: An autonomous web agent for automatic retrieval and identification of interesting publications. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 116–123. ACM Press, May 1998.
3. P. Boncz and M. L. Kersten. Monet: An impressionist sketch of an advanced database system. In *Proc. IEEE BIWIT Workshop, San Sebastian (Spain)*, July 1995.
4. S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your mediators need data conversion! In *ACM SIGMOD*, pages 177–188, 1998.
5. World Wide Web Consortium. *WWW Specifications*. http://www.w3.org/.
6. H. Ehler. Efficient recognition of context-free languages without look-ahead. In Manfred Broy, editor, *Informatik und Mathematik*, pages 230–248. Springer, 1991.
7. D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the world wide web: A survey. *ACM SIGMOD Record*, 27(3):59–74, 1998.
8. S. Ginsburg and M. A. Harrison. Bracketed context-free grammars. *Journal of Computer and System Sciences*, 1(1):1–23, 1967.
9. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
10. R. Jain. Visual information management—introduction. *Communications of the ACM*, 40(12):31–32, December 1997.
11. S. C. Johnson. Yacc: Yet another compiler-compiler. Technical report, Bell Laboratories. Computing Science, 1978.
12. M. L. Kersten, N. Nes, and M. Windhouwer. A feature database for multimedia objects. In *ERCIM Database Research Group Workshop on Metadata for Web Databases*, pages 49–57, Bonn, Germany, 1998.
13. W. R. LaLonde. Regular right part grammars and their parsers. *Communications of the ACM*, 20(10):731–741, 1977.
14. Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based

on declarative specifications. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 132–141. IEEE Computer Society, 1996.