Building the Tower of Babel: Converting XML Documents to VoiceXML for Accessibility[1]

G. Gupta, O. El Khatib, M. F. Noamany, H. Guo

Laboratory for Logic, Databases, and Advanced Programming
Department of Computer Science
New Mexico State University
Las Cruces, NM 88003

Abstract

A recent innovation in the arena of World Wide Web is XML (eXtensible Markup Language). A web-document that has been marked up using an extensible mark-up language (rather than HTML) can be automatically processed on a computer more easily. One XML of great significance to the visually impaired community is VoiceXML, an XML for marking up voice data. A VoiceXML document on the Internet is accessible via the phone. This has the potential of making the Internet universally accessible, provided every document on the WEB is either coded in VoiceXML or is translated to VoiceXML. In this paper, we propose an approach for rapidly translating a document coded in an arbitrary XML to VoiceXML.

1 XML and Marking up Voice Data

Hyper Text Markup Language (HTML) is the language popularly used for marking up information on the World Wide Web so that it can be displayed using commercially available browsers (e.g., Netscape and Internet Explorer). People involved in the development of HTML (the W3 Consortium) have realized that HTML is a weak mark-up language if a user wants to process WEB-pages further. The problem with HTML is that to infer any kind of semantic information *automatically* requires that one examine the content of the web-page and analyze it. The HTML tags do not provide any help, beyond how the document is structured. Thus, writing programs that will further process information contained in the WEB page at the user sites is extremely difficult. This limits the use of information that is available on the WEB. To alleviate this problem, the W3C has come up with an eXtensible Markup Language (or XML) for marking-up information in the WEB pages. XML is indeed flexible in that it allows developers/writers of WEB pages to define their own tags. These tags can have some well-defined meanings that others can understand. W3C expects that groups of users will evolve their own XML for their particular needs. For example, toy store owners and manufacturers will evolve their own XML that will facilitate the selling, distribution and description of toys. This has already started to happen and a plethora of XML based mark-up languages have appeared. These include: BioML for describing biopolymer sequence information, MathML for marking up math for display on the web, SMIL: an XML for describing multimedia presentations for the Web, CML: an XML for describing chemical information. A partial list of the numerous XMLs being proposed by various industry groups, research organizations, and the Government, can be seen at `http://www.xml.org/xmlorg_catalog.htm`.

[1]This work has been supported by NSF HRD 9800209 and INT 9904063.

2 Universally Accessible WEB: VoiceXML and VoxML

The most significant development, as far as the visually impaired community is concerned is the development of VoxML and VoiceXML, two XML based languages for marking up voice data. The goal behind VoxML and VoiceXML is to make the Internet accessible through the phone. VoxML is being developed by Motorola (http://voxml.mot.com), while VoiceXML is being developed by a consortium of IBM, Lucent, Motorola and AT&T. VoiceXML and VoxML require a voice browser (being developed by designers of VoiceXML and VoxML) that understands VoiceXML or VoxML and that reads out the information contained in the VoiceXML or VoxML marked-up web page. Both VoiceXML and VoxML also allow the user to interact with the web-page. Of course, these companies are developing VoiceXML and VoxML with the intent of making the web-pages and the Internet accessible via the phone (or access the WEB in a car while driving), and in the process doing a great service to the visually-impaired community. Thus, if all documents in the WEB were written in VoiceXML or VoxML, they will be accessible to the visually impaired community as well. However, this is not so, since given the tens (or hundreds) of XML-based languages that are being designed, only some documents will be coded in VoiceXML. Most will be coded using the XML the document most closely relates to (for example, all legal documents will be perhaps marked up with Legal XML, rather than with VoiceXML). Thus, while XML is going to allow web-page writers to pack more semantic information about the WEB pages, it is also going to lead to a plethora of XMLs each of which is different from the other, leading to the following problem: WEB-pages prepared by one community (that uses its own XML to prepare it) cannot be understood by another community, because this other community has its own language. This leads to the compromise of the dream of building a seamless Internet, just as in the biblical story of the tower of Babel where the dream of the people to build a tower that reaches the heaven was compromised when God created a plethora of spoken languages and people could no longer communicate with each other.

3 Making XML documents Universally Accessible

In this paper we propose an approach to make XMLs of all types universally accessible. Our idea is to develop a methodology that will allow us to *rapidly* build translation filter software to automatically translate one type of XML into another. Our main interest is in using this framework to translate a document written in a particular type of XML automatically to VoiceXML or VoxML, thus making that document, that was otherwise inaccessible, accessible to visually impaired individuals. It is important that the methodology that we devise allows for rapid development of the translator software for automatically translating XML to VoiceXML or VoxML. This is because the number of XMLs is probably going to be in the 100s, and if it takes a long time to build these automatic translators, they will never be built. It should be noted that our approach is quite general and can be used for making any document accessible to visually impaired. Essentially, using our techniques any formal documents written using any mark-up language can be made accessible to the visually blind by automatically translating them to VoiceXML.

To build a translator to translate one formal notation to another, traditional compiler technology [8] is normally used. However, a solution based on traditional compiler technology will take a long time, as building translator in such a way will be quite cumbersome. Instead we propose a novel approach that is based on denotational semantics [3] and logic programming [2] that allows one to develop a translator in a very short period of time [7].

Denotational Semantics [3] is a well-established methodology for design and analysis of computer

languages. In the denotational approach, the semantics or meaning of a computer language/notation is specified in terms of mathematical objects (such as sets and functions). A denotational specification has three components: *syntax specification* which maps sentences of the language to parse trees; *semantic algebras* that specify these mathematical objects, and *valuation functions* that map parse trees to semantic algebras. It turns out [7] that the syntax specification, the semantic algebra, and the valuation functions can be specified using logic programming [2]. This logic programming specification is also executable [7]. To obtain a translator from a language $\mathcal{L}_1$ to $\mathcal{L}_2$ we choose the semantic algebra consisting of parse trees of $\mathcal{L}_2$. The logic programming specification then automatically turns into a translator which can be executed to translate sentences of language $\mathcal{L}_1$ to sentences of language $\mathcal{L}_2$.

In our case, the language $\mathcal{L}_1$ will be the XML we want to translate for universal accessibility, while the language $\mathcal{L}_2$ will be VoiceXML or VoxML. Using the technology of denotational semantics and logic programming, the translation filters can be developed very rapidly. Consider an example where we want to make MathML (an XML for coding Mathematics) accessible to visually impaired students. Essentially, using the denotational approach, the syntax of MathML will be expressed using the Definite Clause Grammar facility [2] of logic programming. In fact, the DCG can be automatically generated from the Document Type Definition (DTD) of an XML document. Next, the semantics of MathML will be given in terms of VoiceXML or VoxML constructs. Both the syntax specification and the semantics specification will be expressed using logic programming. The logic programming specification of syntax and semantics is executable, yielding a parser automatically.

Our methodology, based on logic programming and denotational semantics, will permit individuals to very rapidly construct a filter between any type of XML to VoxML or VoiceXML, making XML documents universally accessible. Our methodology can also be used for rapid construction of a translator between one type of XML to another, for example, to translate BioML documents to documents in CML (Chemical Markup Language). It can also be used to translate HTML to VoiceXML, thus making HTML-coded web-pages accessible as well, as is illustrated next.

4 An Example: Converting HTML to VoiceXML

To illustrate our approach, we show how a subset of HTML can be translated to VoiceXML. To obtain such a translator one can specify the semantics of HTML text in terms of VoiceXML. We have chosen a subset of the HTML that is suited to VoiceXML translation, including the radio button, the select, the head tags and the title tag. For instance, we want to translate HTML document shown to the left to a VoiceXML document shown to the right below.

```
% HTML program
<html>
<head>
<title> Menu </title>
</head>
<body>
Menu
<form>
<select name=MenuApp>
<option> Chicken </option>
<option> Meat </option>
<option> Fish </option>
<option> Vegetable Salad </option>
</select>
```

```
% VoiceXML
<?xml version=1.0>
<vxml>
<form>
<block> Page Title:  Menu  </block>
<block> Menu  </block>
<field name=MenuApp>
<prompt>  Vegetable Salad  OR  Fish  OR
          Meat  OR  Chicken </prompt>
<grammar>  Vegetable Salad  Fish  Meat
           Chicken </grammar>
</field>
<block> Choose your desert : </block>
<field name=omar>
```

```
</form>                                          <prompt> Ice Cream OR Apple Cake</prompt>
<form>                                           <grammar> Ice CreamApple Cake</grammar>
Choose your desert : Ice Cream                   </field>
<input type=radio name=omar value="ddd">         </form>
Apple Cake                                       </vxml>
<input type=radio name=omar value="ddd">
</form>
</body>
</html>
```

Since HTML has a well-formed context free grammars, it is very natural and easy to write the DCG grammar as shown below, where almost one DCG rule corresponds to one BNF grammar rule. The building of the parse tree is also very easily specified, by adding an extra argument in which the parse tree is synthesized. The parse tree of a phrase is constructed in terms of the parse trees of its component sub-phrases. The DCG rules above constitute an executable program, as they will be translated into ordinary Prolog clauses automatically when they are loaded into the Prolog system. Thus, one can completely avoid having to deal with the algorithmic or computational aspects of parsing. Definite Clause Grammars allow context-free grammars to be easily expressed in Prolog, and the grammar specification automatically acts as a parser. A parser for a subset of HTML as a DCG is shown below.

```
html_document(html_document(X)) -->  html_tag(X) .
html_tag(html_tag(X)) -->  ['<html>'], html_content(X), ['</html>'].
html_content(ht(X,Y)) -->  head_tag(X), body_tag(Y).

head_tag(head_tag(X)) -->  ['<head>'], title_tag(X), ['</head>'].
title_tag(title_tag(X)) -->  ['<title>'], plain_text(X), ['</title]>'].

body_tag(body_tag(X)) -->  ['<body>'], xbody_content(X),['</body>'].
xbody_content(xb(X,Y)) --> body_content(X), xbody_content(Y).
xbody_content(xb(X)) --> body_content(X).
body_content(X) --> text(X).
boby_content(X) --> form_tag(X).

form_tag(form_tag(X)) -->  ['<form>'], xform_content(X), ['</form>'].
xform_content(xf(X,Y)) --> form_content(X) ,xform_content(Y).
xform_content(xf(X)) --> form_content(X).

form_content(input_radio(X, Y)) --> plain_text(X), [:], xradio(Y).
form_content(select_option(X,Y)) --> [<],[select],[name],[=],string(X),[>],
    xoption(Y),['</select>'].

xradio(xradio(X, Y)) --> radio(X), xradion(Y).
xradio(X) --> radio(X).
radio(radio(X, Y, Z)) --> plain_text(X), [<],[input],[type],[=],[radio],
     [name],[=],string(Y),[value],[=], [#],string(Z),[#],[>].

xoption(xoption(X, Y)) --> option(X), xoption(X, Y).
xoption(X) --> option(X).
option(option(Y)) --> ['<option>'], plain_text(Y),['</option>'].
```

In case of XML, one can do even better and generate a parser automatically from its Document Type Definition (DTD). The syntax structure of an XML is given by specifying its Document Type

Definition [4] (DTD). The DTD of an XML specifies the structure of the documents marked-up using that particular XML. The DTD is specified in the extended BNF format in a special metalanguage designed by the W3 consortium. One can use our approach to translation based on logic programming and denotational semantics to give the semantics of a DTD expressed in this meta-language in terms of its corresponding DCG grammar. This semantics, which is executable, can then be used to automatically generate a DCG grammar from a given DTD. The DCG grammar, when run on a logic programming system, automatically supplies a parser for that DTD.

Semantic interpretation is the second step of the denotational semantics-based approach, in which the mapping functions (called *valuation predicates*) from HTML parse trees to the corresponding semantic values in terms of VoiceXML parse trees are specified. These semantic function are specified as logic programs in our approach:

```
to_voice(html_document(X), T) :- shtml(X, T1),
    append([['<?xml version=1.0>'], ['<vxml>'], ['<form>']], T1, T2),
    append(T2, [['</form>'], ['</vxml>']], T).
shtml(html_tag(X), T) :- sht(X, T).
sht(ht(H, B), T) :- shead(H, T1), sbody(B, T2), append(T1, T2, T).
shead(head_tag(X), T) :- shead_tag(X, T).
sbody(body_tag(X), T) :- sbody_tag(X, T).

shead_tag(head_content(X), [['<block> page Title:', T1, '</block>']]) :-
    shead_content(X, T1).
shead_content(title_tag(X), X).
sbody_tag(xb(X, Y), T) :- sxb(X, T1), sbody_tag(Y, T2), append(T1, T2, T).
sbody_tag(xb(X), T) :- sxb(X, T).

sxb(text(X), [['<block>', X, '</block>']]).
sxb(form_tag(X), T) :- sform_tag(X, T).
sform_tag(xf(X, Y), T) :- sxf(X, T1), sform_tag(Y, T2), append(T1, T2, T).
sform_tag(xf(X), T) :- sxf(X, T).

sxf(select_option(X, Y), [['<field name =', X, '>'], T3, T4, ['</field>']]) :-
    sxoption(Y, T1, T2), append(['<prompt>' | T1], ['</prompt>'], T3),
    append(['<grammar>' | T2], ['</grammar>'], T4).

sxf(input_radio(X, Y), T) :-
    sxradio(Y, T1, T2, N), append(['<prompt>' | T1], ['</prompt>'], T3),
    append(['<grammar>' | T2], ['</grammar>'], T4),
    append([['<block>', X, ':', '</block>'], ['<field name =', N]], [T3], T5),
    append(T5, [T4, ['</field>']], T).

sxoption(xoption(X, Y), T, TT) :- soption(X, T1), sxoption(Y, T2),
    append(T1, ['OR'|T2], T), append(T1, ['\|' | T2], TT).
sxoption(option(X), [X], [X]).

sxradio(radio(X, Y), T, TT, N) :-
    sradio(X, T1, N), sxradio(Y, T2, TT2, N),
    append(T1, ['OR'|T2], T), append(T1, ['\|' | TT2], TT).
sradio(radio(X, Y, _), [X], Y).
```

Note that there is almost one Prolog valuation predicate per DCG grammar rule. The valuation function assigns a meaning to the syntax tree based on the meanings assigned to its subtrees. Since

we have defined the mappings of the valuation functions on all of the options listed for the DCG rules above, the valuation functions have been completely defined for the subset of HTML considered.

Once the semantic specification is complete, it can be loaded in a Prolog system and used to convert parse trees of HTML to VoiceXML documents. The HTML parse tree for an HTML document can be created using the HTML syntax specification expressed as a DCG that can also be run on a Prolog system. Thus, together the syntax and semantic specification provide a complete translator from HTML to VoiceXML.

In the example above, we show how our approach can be used to convert HTML documents to VoiceXML documents. In general, a document marked up using any type of XML can be automatically converted to a VoiceXML document or to a document marked-up using a different type of XML.

5 Related Work and Conclusions

The W3 consortium has come up with *XSLT (XSL Transformation)* [6], a language for specifying transformation of one XML document to another. XSLT is part of *XSL* (eXtensible Stylesheet Language) [5] which is a language for indicating how an XML document is to be formatted (e.g., for rendering on a web-browser). Thus, a mapping from one XML document to another XML can also be specified as an XSLT specification. However, the XSLT approach is limited, since, in the words of designers of XSLT "XSLT is not intended as a completely general-purpose XML transformation language. Rather it is designed primarily for the kinds of transformations that are needed when XSLT is used as part of XSL." In contrast, due to the generality of logic programming, our approach allows one to transform an XML document in any arbitrary way. Also, note that XSLT is a transformation specification language. Our approach can be viewed as one way of rapidly implementing the XSLT.

In this paper we presented an approach to transforming documents written in XML to VoiceXML so that they become universally accessible. Our approach is based on the technology of logic programming and denotational semantics. Our approach is quite general purpose and can also be applied to other types of transformation tasks [1].

References

[1] A. Karshmer, G. Gupta, S. Geiger, C. Weaver. A Framework for Translation of Braille Nemeth Code to LaTeX: The MAVIS Project. In *Proc. ACM Conference on Assistive Technologies*, ACM Press, pp. 136-143, Mar. 1998.

[2] L. Sterling & S. Shapiro. The Art of Prolog. MIT Press, '94.

[3] D. Schmidt. *Denotational Semantics: a Methodology for Language Development*. W.C. Brown Publishers, 1986.

[4] http://www.xml.org.

[5] http://www.w3.org/Style/XSL.

[6] http://www.w3.org/TR/xslt.

[7] G. Gupta. Horn Logic Denotations and Their Applications. In *The Logic Programming Paradigm: A 25 year perspective*. Springer Verlag. pp. 127-160. May 1999.

[8] A. Aho, J.D. Ullman, R. Sethi. Compilers: Principles, Techniques, and Tools. Addison Wesley. 1986.