Disassembly knowledge representation via XML

Ibrahim Zeid and Surendra M. Gupta
Laboratory for Responsible Manufacturing
334 SN, Department of MIME
Northeastern University
360 Huntington Avenue
Boston, MA 02115

ABSTRACT

Extensible markup language (XML) is a new powerful technique for Web and Internet development. It is a method of defining structured data in a text file. XML is expected to do for data what HTML has done for Web pages. XML's strength lies in its simplicity to represent data and knowledge. It can maintain hierarchical structures encountered in many systems including assemblies and disassemblies. It can also be well integrated with Java, and its Java beans. Its strength lies in its flexibility to adapt to any knowledge domain because it is a metalanguage. It is used to develop modeling languages that are tailored to specific knowledge, and specific data structures and hierarchies. This paper presents an overview of XML, followed by a proposal of an XML-based knowledge representation model for disassembly planning. An example is used to demonstrate the capabilities of the proposed XML model.

Keywords: Design and planning for disassembly, HTML, XML, Internet and the Web, Java and Java beans.

1. INTRODUCTION

HTML is the most frequently used language on the Web. The reason for its popularity is its simplicity. HTML defines a simple and fixed type of a document that is formatted via a fixed set of tags[1]. A browser renders the tags and displays the page. HTML is most suited for Web pages with headings, paragraphs, links, lists, images, sound, video, forms, and tables. This is what HTML is set and designed to do, in the first place. Web pages belong to a common class of office or technical reports. However, many innovative Web authors and vendors of browsers have extended HTML as far as possible to accommodate other needs. As a result, many extensions to HTML tags and attributes, and many layers of functionality and capability have been created. These extensions and layers have, to some extent, obscured the elegant basic framework that HTML defines.

As useful as HTML is, it is limited and difficult to extend to other types of documents. First, it consists of a finite pre-defined set of tags. Second, HTML does not provide a semantic context to the data used. Third, HTML does not support complex data organizations or object hierarchies. These hierarchies are found in database applications, or scientific and engineering (software and others) fields. XML overcomes all three HTML shortcomings. XML is extensible, allowing users to define their own tags with their own attributes and values. These values provide semantic qualifications to data and context. XML provides context validation at both the semantic and syntactical levels. It checks for completeness and well formedness of documents. XML also supports rich structures, similar to those found in object oriented programming, and database applications.

XML is a metalanguage[2]. XML is simply a set of rules for creating new document types. XML can be used to construct a new markup language for a specific Web application, such as studying chemistry molecules, or formatting mathematical equations. These new markup languages are known as XML dialects, or XML component grammars.

2. XML DOCUMENTS

XML defines XML documents. XML documents define data objects. XML is based on the well-known object oriented paradigm. XML documents provide access to the content and structure of the data objects. An XML document, like an HTML document, uses tags to define both the data and structure of the objects that the document defines, and eventually creates. The structure of the data defines the object hierarchy. As in HTML, XML tags can be nested. They may have attributes. Attributes may have values. However, there are some differences between HTML and XML tags. XML tags

follow much stricter rules. For example, XML tags must be closed. They are case sensitive, whitespace matters, and all attribute values must be enclosed in quotes.

Let us consider developing an XML document to describe a car. To write the XML document of the properties of a car, we need to invent our own custom markup language for describing cars. We describe the information structure of the car properties, and how the different pieces of data and information relate to each other. Let us assume that the car has properties, and instructions. As a result, the XML code is listed below, and the XML page is shown in figure 1 using MS IE 5.0. Figure 2 shows the equivalent HTML Web page of the car's XML document in Figure 1.

```
<?xml version="1.0" standalone="yes"?>
<Car>
  <Name>The best car</Name>
  <Description>Here are the properties of the best car.</Description>
  <Properties>
        <Property>
           <Item>Body</Item>
           <Quantity units="Ib">2000</Quantity>
        </Property>
        <Property>
           <Item>Tires</Item>
           <Quantity>4</Quantity>
        </Property>
        <Property>
           <Item>Engine</Item>
           <Quantity units="cylinder">4</Quantity>
        </Property>
  </Properties>
  <Instructions>
        <Step>Read owner's manual</Step>
        <Step>Change oil</Step>
        <Step>Rotate tires</Step>
  </Instructions>
</Car>
```

The XML code shown above is well formed according to the XML rules[2]. The XML document describes the information about the car in terms of its properties. The semantics of the data are maintained in the XML document.

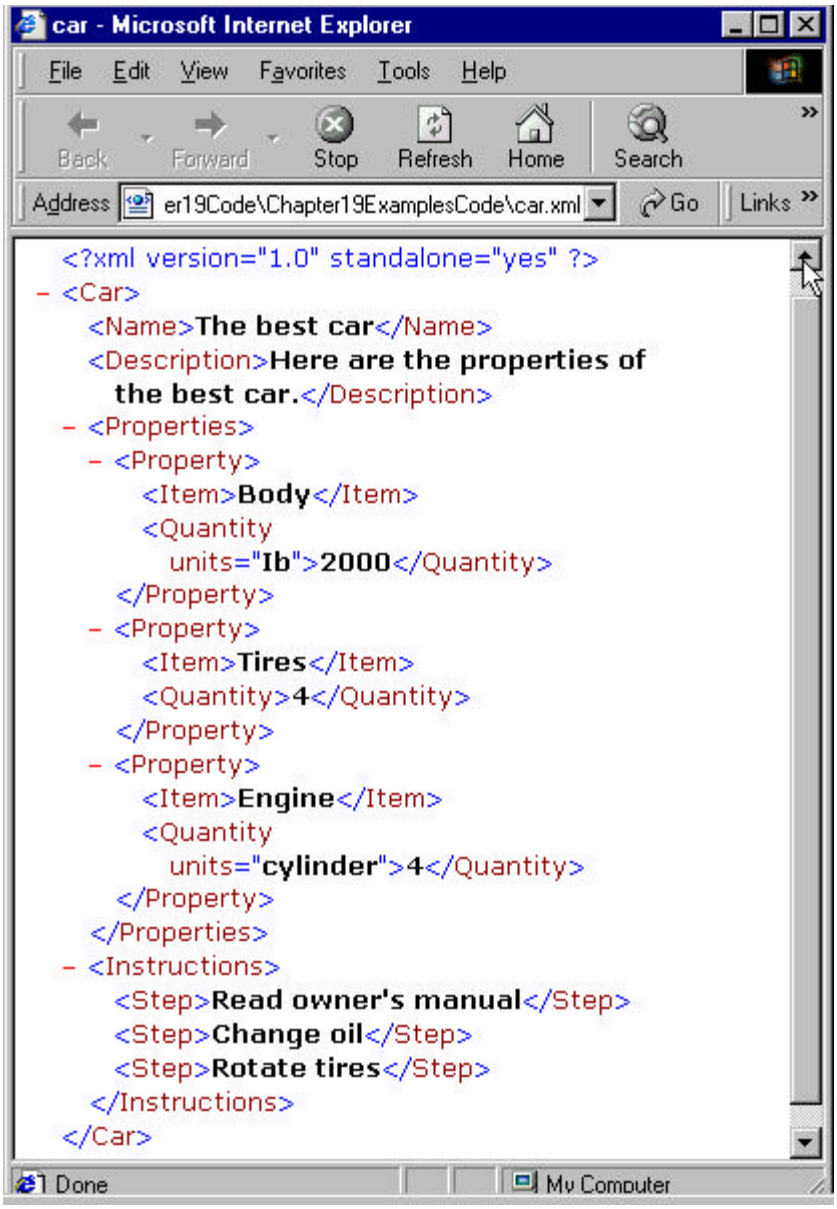

Figure 1. XML hierarchical structure of a car

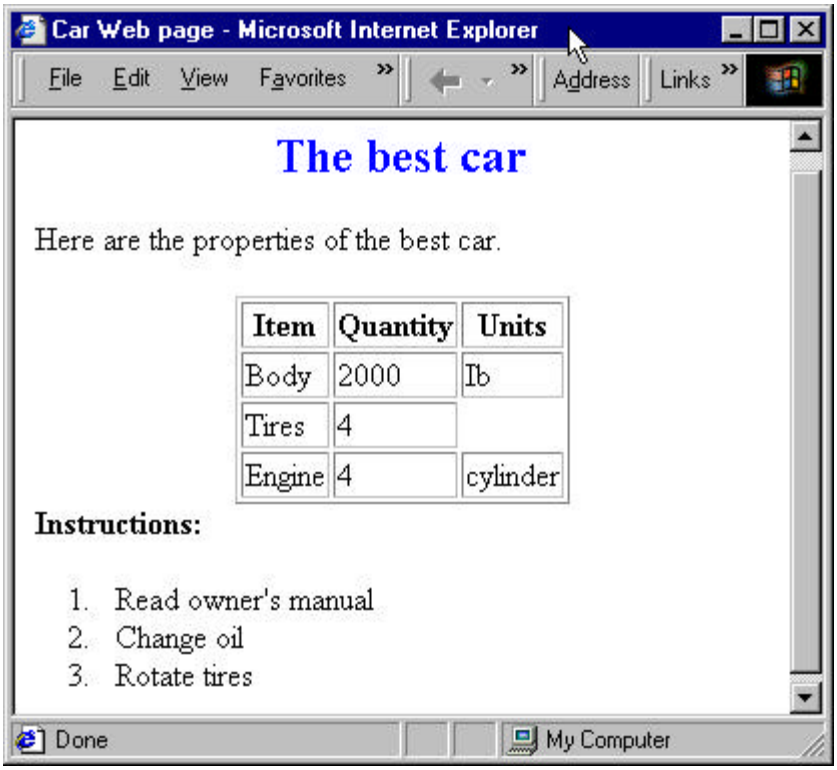

Figure 2. The equivalent HTML Web page of the car

The above XML code has four hierarchies or levels. At the top level is the <Car> tag. This tag encloses all the other tags. The second level has the <Name>, <Description>, <Properties>, and <Instructions> tags. The <Name> and <Description> tags serve the role of the HTML text. The <Properties> tag acts as a container for the car properties. The <Instructions> tag replaces the HTML <OL> tag. The third level has the <Property> and <Step> tags. There is one <Property> tag for each of the three car properties we have. There are three <Step> tags for the three instructions. The fourth and the last level has the <Item> and <Quantity> tags. We assume that each car property is expressed by an item and its quantity. The <Quantity> tag has the units attribute.

3. LOGICAL STRUCTURE

The logical structure of an XML document consists of declarations, tags (elements), comments, characters, and processing instructions. All of this logical structure is indicated in the document by explicit markup as shown in the XML car example we have presented. The logical structure of an XML document refers to its tags, their structures relative to each other, and their semantics. Tags come in two groups. One group has tags that start and end. The other group has empty tags. The logical structure must be well formed and valid. The well formedness and validity requirements are used by XML parsers (processors) during processing XML documents.

It is much easier to check the well formedness of a document than checking its validity. All what we need to do to check if a document is well formed, is to check its tags against XML syntax rules. Checking the document validity requires us to establish the semantics rules that we use as a reference to measure the document semantics against. These semantics are included in the DTD (document type definition). Thus, we need to write a DTD for every XML markup language we create. DTDs are used by XML parsers to validate XML documents. A DTD contains statements that define to the parser what is possible to do in a valid XML document that uses this DTD. Each statement looks like a tag. XML parsers read all DTD statements before they begin parsing a document.

The DTD[2] for the XML car document shown in Figure 1 is shown below:

<!ELEMENT Car (Name, Description?, Properties?, Instructions?)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Properties (Property)*)>
<!ELEMENT Property (Item, Quantity)>
<!ELEMENT Item (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<ATTLIS Quantity units CDATA>
<!ELEMENT Instructions (Step)+>

4. PHYSICAL STRUCTURE

XML documents could be referring to other documents that they may need. The physical structure of an XML document consists of all entities that are contained in the document. The main document represents the "root" or the "document" entity. The document entity serves as the starting point for an XML parser. Entities could be parsed or not. Parsed entities are replaced by their corresponding text that becomes part of the main XML document. Unparsed entities usually refer to a resource whose contents are not text. For example, an executable that is referenced in an XML document is an unparsed entity.

XML identifies three types of parsed entities: general, character, and parameter. General entities are simply known as entities. They can be used to define shortcuts to substitute for long text. The general format to declare a general entity is <!ENTITY entity_name text_to_replace>. Character entities are special cases of general entities. They refer to the ISO character set. We can define and use a character entity for the Greek symbol ALPHA as follows <!ENTITY ALPHA "&945;">. Parameters entities allow a DTD to access an entire external document and make its content available to the current XML document. The general format for a parameter entity is <!ENTITY % entity_name text_to_replace>.

5. PARSERS AND VIEWERS

The ultimate goal of writing XML documents is to display and use them on the Web. Rendering XML documents requires more work than rendering HTML documents. XML tools must check both the syntax and semantics of an XML document before displaying it. We can identify these tools as parsers and viewers. XML parsers check both the syntax and the semantics. XML viewers display the XML document. Parsers and viewers can come in separate software, or may be bundled together.

There are two types of parsers: validating and non-validating. If the parser only checks the document syntax, then it is non-validating. This type of a parser only checks the well formedness of an XML document. Non-validating parsers do not use the DTD of the XML document they are parsing. Validating parsers, on the other hand, checks both the well formedness and validity of the XML document. They check both the syntax and semantics of XML documents. These parsers use DTDs while performing the validity checks. They are also harder to develop and write than non-validating parsers.

Once the XML document is parsed and/or validated, the parser passes it to an XML viewer to display it. An XML document may be displayed in three formats. The viewer may just display back the XML code, as shown in figure 1. The two other formats that viewers can use to display XML documents are tree structure, or graphics. An XML viewer may display the tree structure of the XML document. This structure shows the hierarchy of the objects used by the document. The viewer may display the XML in graphics format. The content of the XML document is shown with all its text, links, images, and so forth. An XML viewer may offer the three formats of display to the user.

6. XML MODEL

The complete XML model requires two documents: the XML document and the DTD. We pass the two files to a validating XML parser that parses the XML file using the DTD file. If there are syntax or semantics errors, we correct the documents. When the parsing process is successful, the parser may create a data object model (DOM). This model has the object tree of the XML document. We use a viewer to display the XML document.

The DOM that XML uses is similar to object models used by Internet tools such as dynamic HTML that is commonly implemented in Web pages via JavaScript. DOM provides a general way of accessing data structures and objects from structured documents. Thus, an application can use XML to manipulate data structures and objects.

7. APPLYING XML TO DISASSEMBLY PLANNING

The first step in using XML in disassembly planning is to be able to represent disassembly knowledge via XML. Case-based reasoning (CBR) [3, 4, 5] has been applied to disassembly planning[6 7] by the authors[8, 9] (For a more detailed review of the literature on disassembly, recycling and product recovery, see Gungor and Gupta[10]. Moyer and Gupta[11] present a state-of-the-art review on operational issues of disassembly and recycling in the electronics industry). The CBR technique is based on using EMOP memory model (tree structure) where EMOPs form the tree nodes, while the tree leaves represent data. In disassembly context, the EMOPs represent decision points, such as an assembly, subassembly, or a component. The leaves represent disassembly plans. Thus, the EMOP tree structure holds both the products structure and the disassembly sequence.

We can represent the same disassembly knowledge using XML instead of using case-based reasoning and its EMOP memory model. Once the representation is complete, we manipulate the resulting knowledge base to retrieve previous disassembly plans. Suppose that the structure of a lamp is presented as shown in figure 3.

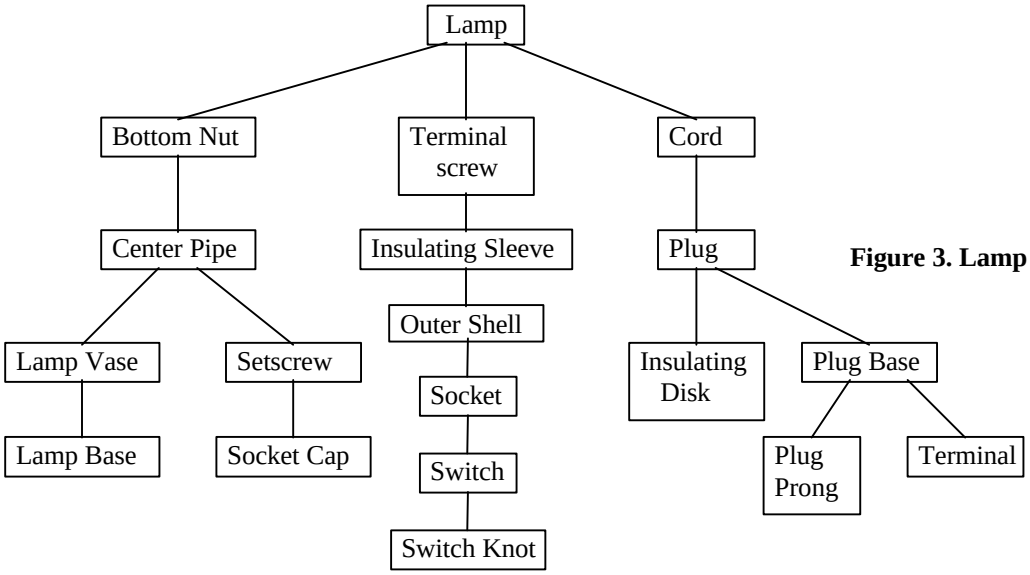

Figure 3. Lamp Structure Tree

This structure can be captured in XML via an XML document and a DTD document. The XML document is as follows:

```
<?xml version="1.0" standalone="yes"?>
<Lamp>
  <Name>Lamp structure</Name>
  <Description>Lamp disassembly tree</Description>
  <Bottom Nut>
        <Center Pipe>
              <Lamp Vase>
              </Lamp Vase>
              <Lamp Base>
              </Lamp Base>
              <Setscrew>
              </Setscrew>
              < Socket Cap>
              </ Socket Cap>
```

```
        </Center Pipe>
    </Bottom Nut>
    <Terminal screw>
    .
    .
    </Terminal screw>
    <Cord>
    .
    .
    </Cord>
</Lamp>
```

The DTD document is as follows:

```
<!ELEMENT Lamp (Name, Description?, Bottom Nut, Terminal screw, Cord)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Bottom Nut (Center Pipe)*)>
<!ELEMENT Center Pipe (Lamp Vase, Lamp Base, Setscrew, Socket Cap)>
<!ELEMENT Lamp Vase (#PCDATA)>
<!ELEMENT Lamp Base (#PCDATA)>
<!ELEMENT Setscrew (#PCDATA)>
<!ELEMENT Socket Cap (#PCDATA)>
```

Inside each of the above XML elements, we can add disassembly plans, as well as any other disassembly algorithms to calculate such items as disassembly time, cost, etc.

8. CONCLUSIONS

XML has been introduced and proposed in this paper as a new and novel technique to represent disassembly knowledge. This technique seems superior to previous techniques such as EMOP memory models. The technique will also allow developing Web-based disassembly methodologies. Further research is still needed to fully implement and develop the proposed disassembly XML model.

REFERENCES

1. Zeid, I., *Mastering the Internet and HTML*, Prentice-Hall, 2000.
2. Bradley, N., *The XML companion*, Addison-Wesley, 2000.
3. Schank, R.C., *Dynamic Memory*, Cambridge University Press, Cambridge1982.
4. Schank, R.C. and Adelson, R.P., *Scripts, Plans, Goals and Understanding*, Lawrence Erlebaum, Hillsdale, NJ, 1977.
5. Kolodner, J. L. *Retrieval and organizational strategies in conceptual memory: A computer model*, Lawrence Erlbaum, Hillsdale, NJ, 1984.
6. Brennan, L., Gupta, S. M. and Taleb, K. N., “Operations planning Issues in an Assembly/Disassembly Environment”, *International Journal of Operations and Production Management*, Vol. 14, No.9, 57-67, 1994.
7. Gupta, S. M. and Taleb, K. N., “Scheduling Disassembly”, *International Journal of Production Research*, Vol. 32, No. 8, 1857-1866, 1994.
8. Zeid, I., Gupta, S. M. and Bardasz, T., "A case-based reasoning approach to planning for disassembly", *Journal of Intelligent Manufacturing,* Vol. 8, No. 2, 97-106, 1997.
9. Gupta, S. M. and Veerakamolmal, P., “A Case-Based Reasoning Approach for Optimal Planning of Multi-Product/ Multi-Manufacturer Disassembly Processes”, *International Journal of Environmentally Conscious Design and Manufacturing*, Vol. 9, No. 1, 15-25, 2000.
10. Gungor, A. and Gupta, S. M., “Issues in Environmentally Conscious Manufacturing and Product Recovery: A Survey”, *Computers and Industrial Engineering*, Vol. 36, No. 4, 811-853, 1999.
11. Moyer, L. and Gupta, S. M., "Environmental Concerns and Recycling/ Disassembly Efforts in the Electronics Industry", *Journal of Electronics Manufacturing*, Vol. 7, No. 1, 1-22, 1997.