USING XML FOR EDI

by
Edwin E. Van Duinen

A Master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Master of Science in
Information Science.

Chapel Hill, North Carolina

July, 1999

Approved by:

_____

Advisor

Edwin E. Van Duinen. Using XML for EDI. A Master's paper for the M.S. in I.S. degree. July, 1999. 45 pages. Advisor: Gregory B. Newby

This paper examines XML and Electronic Data Interchange (EDI) to see how XML can be used for EDI. The paper is concerned with the ability of XML to actually provides solutions to EDI's adoption barriers. An overview of XML's components and constraints is presented. Next, EDI is addressed, paying particular attention to its components and the barriers to EDI adoption. Finally, a discussion of how XML can be used for EDI occurs with focus also given to the impact of XML on EDI's adoption barriers. This paper concludes that if properly implemented, XML does offer solutions to EDI's problems. Sources include published literature, publicly accessible web pages, and standards body produced specifications.

Headings:

XML (Document markup language)

Electronic data interchange

Table of Contents

Introduction

Electronic commerce is a concept concerned with all facets of business transactions conducted through electronic means. Electronic commerce can roughly be divided into two major sectors: business-to-business and business-to-consumer. The first sector involves business conducted between two organizations. The later involves business between an organization and an individual.

Business-to-business electronic commerce differs from business-to-consumer electronic commerce in its fundamental makeup. For instance, when businesses purchase a product, they often issue requests for quotations, receive quotations, issue purchase orders, receive invoices, and provide payment after receipt of goods. Consumers, on the other hand, often select products with a set price from a catalog, provide payment, and then receive the goods. These subtle differences introduce distinct processing requirements and system components between business-to-business and business-to-consumer electronic commerce solutions.

The overall business-to-business electronic commerce market is expected to expand from $43 billion to over $1.3 trillion in the next few years ("Business-to-Business E-Commerce", 1999). Electronic Data Interchange (EDI) is concerned with the business-to-business sector of electronic commerce. It provides a method for two autonomous computer systems to exchange standardized business documents. EDI has traditionally

made up the bulk of all of electronic commerce. However, EDI growth is actually expected to lessen over this time period. In fact, the Gartner Group predicts EDI growth will drop from nearly 30% to less than 15% (Shih & Terhune, 1998).

Alternative business-to-business electronic commerce solutions are beginning to flourish. Perhaps this signals the impending demise of EDI. However, EDI is not dead yet. Many modifications to EDI are appearing, but none seem as promising as those employing XML technologies do. How can XML be used for EDI? More importantly, does XML actually present a solution to EDI's adoption barriers?

The first section of this paper explores XML. Next, EDI is investigated to determine what components make up an EDI system and what the barriers to EDI adoption are. The third section addresses the use of XML for EDI. Finally, conclusions of this paper are presented.

What is XML?

XML is the Extensible Markup Language. It was formally ratified as a W3C standard on February 10, 1998. The specification provides a set of grammar and syntax rules for semantically describing the structure of data. The specification also provides a limited description of the behavior of processing computer programs (World Wide Web Consortium [W3C], 1998a).

XML's roots lie in SGML and HTML. The Standard Generalized Markup Language (SGML) is an ISO standard which allows one to describe data's semantic meaning and structure through markup. Like XML, SGML is a metalanguage. SGML is very flexible and powerful, but also extremely complex. It is this complexity which often makes SGML too cumbersome and expensive to use.

HTML is an application of SGML originally designed to describe scientific papers. HTML is simple to use and allows for the easy exchange of documents through hypertext links. The mid-1990's explosion of the World Wide Web found HTML being (mis)used for page layout/stylistic purposes as well as for shoehorning other types of data into its limited tag set. Simply put, HTML was being used for purposes for which it was not designed.

The architects of the World Wide Web found the need for a new language which combined SGML's flexibility and power with HTML's simplicity and Internet aware

features. They began work on such a language in 1996. The result is XML.

XML is a metalanguage which allows other languages to be formally defined. The resulting languages are "XML applications" (Harold, 1998, p.16). XML, the metalangauge, is not an application unto itself. Instead, XML can be seen as the foundation upon which a wide variety of applications dealing with structured data can be built (W3C, 1999a).

In a nutshell, XML allows one to encode, or markup, text documents with a tag set and structure that the author defines. This ability to define the document's tags and structure is what makes XML extensible. Appropriately named tags give semantic meaning to the encoded content. Furthermore, hierarchical combinations of tags give structure to the data. The ability to add structure to the data makes it possible to use XML encoded data for a wide variety of applications.

Components of XML Documents

XML documents can be of two types: well-formed and valid. Well-formed documents are merely syntactically correct. Valid documents are well-formed documents which also adhere to a defined structure.

Physically speaking, XML data is stored in units called entities. Entities may contain text or binary data and may reside inside or outside of the XML document proper. However, entities stored inside the XML document may only include Unicode text. (ASCII text is a subset of Unicode and is therefore permitted.)

From a logical perspective, XML documents can be divided into two major

sections: the prolog and the "root" element. The only truly required section of the XML document is the "root" element which will be described in the next section.

The "Root" Element

The "root" element contains the document's data. Various combinations of the following components may be combined to makeup the "root" element's (or any element's) content:

- Elements
- Text
- CDATA
- Entity References
- Comments
- Processing Instructions

Elements

The content of XML documents essentially contains data marked up with user defined tags. In XML, these tags are referred to as elements. An element in XML consists of a start-tag/end-tag pair, along with the enclosed content.

XML elements are given names by the document author. Element names should be chosen so as to adequately describe the element's contents. Furthermore, elements can contain other elements to create structure. These traits provide the ability to construct documents with a semantically understandable structure and content. This should make it

easier for humans to write programs which process the data enclosed in XML document structures.

Element tags are delimited by the < and > characters and immediately begin with the element's name. The end-tag begins with a "/" before the element name. "Whitespace" in XML documents is usually ignored. The following is an example of an element:

```
<myElement>
    Here is some enclosed text which makes up the content of
    the element named "myElement".
    <myChildElement>
        This is the content of myChildElement which is
        contained within myElement. Therefore, this content
        is also contained within myElement.
    </myChildElement>
</myElement>
```

While it is syntactically correct to immediately close an element containing no content with an end-tag, XML allows for the "empty" element. An empty element is an element which does not contain any enclosed content. Empty elements consists of a single tag which is delimited by the < and /> characters. Here is an example of an empty element:

```
<myEmptyElement/>
```

Attributes of Elements

Elements, including empty elements, may have attributes associated with them which do not constitute the content of the element (Boumphrey et al., 1998). Attributes may contain character data, uniquely identify an instance of an element, refer to other locations in the document, cause the inclusion of external binary data, or simply have a name value from a defined enumeration. Certain element's attributes may be defined to be required or even given default values. An attribute takes the form of name="value" within

an element's start-tag. For instance:

```
<myEmptyElement attribute1="value1" attribute2="value2"/>
```

Text

Within an XML document, data is stored as text and text is a string of character data. XML makes no distinction between numeric values and character values. To XML, all data or content is stored as a text string. Thus, numeric data stored within an XML document must be converted to and from a text string by the processing application.

Furthermore, text can be intermingled with markup. A unit of text which contains markup can be parsed into the character data portions and the other components. Text is therefore referred to as parseable character data and is notated as #PCDATA.

So that processing applications do not mistake character data as markup delimiters, text data may not contain the markup delimiter characters such as `<` and `&`. Entity references must be used to include these characters in the document's content. Please refer to the following description of entity references.

CDATA

In some cases, text may contain numerous instances of the delimiter characters, but not contain any actual markup. It would be impractical or undesirable to convert all instances of those characters into entity references. XML allows one to store text as character data which by definition may not contain other markup. Such a section of character data is known as CDATA. The beginning and end of CDATA sections are delimited by `<![CDATA[` and `]]>` respectively.

Entity References

Entity references exist in XML to point to entities defined in the document type definition (DTD) (see the section on the DTD for further explanation) or required by the XML specification. There are essentially four types of references: character references, internal entity references, external entity references, and parameter entity references. Processing applications may expand references to include the entity they refer to.

Character References

A character reference refers to a specific character of the Unicode character set. Character references permit certain characters to be included in XML without having the processing application interpret them as special characters. Character references also permit the inclusion of characters not directly accessible by an input device such as a keyboard. For instance, Parseable Character Data in XML may not include the < character. In order to include this character in the data, one would represent it with `<`. Another example would be to use `&#165;` to represent the `&yen;` character which may not be directly accessible from the keyboard. Character references are delimited by the `&` and `;` characters.

Internal Entity References

Internal entity references function much like a simple macro. They allow the author to define, inside the XML document proper, replacement text which will be expanded by the processing application. For instance, `&edi;` may be defined to refer to the entity "Electronic Data Interchange". One could then use `&edi;` in the document

instead of repetitively typing out Electronic Data Interchange. Internal entity references are delimited by the `&` and `;` characters.

External Entity References

External entity references function similarly to internal entity references, except that the entity they refer to is found external to the XML document proper. This also allows for the inclusion of non-textual data as part of the document. External entity references are particularly useful because they permit the inclusion of dynamically generated data from an outside source such as a database system within the document body. External entity references are delimited by the `&` and `;` characters.

Parameter Entity References

Parameter entity references can only be used in the DTD (detailed below) and allow the grouping of components into sets to facilitate more easily maintainable DTDs. For instance, the parameter entity `%pe;` could be defined to include elements `A`, `B`, and `C`. One could then define the element `X`, the element `Y`, and the element `Z` to contain `%pe;` without having to name `A`, `B`, and `C` multiple times. Parameter entity references are delimited by the `%` and `;` characters.

Comments

Comments allow the author to include descriptive notes for those reading the XML code. Comments are text strings delimited by `<!--` and `-->`. Comments may be found nearly anywhere in the XML document, including in the prolog or after the "root"

element.

Processing Instructions

Processing instructions allow messages intended for processing applications to be embedded within the code. For instance, one could embed a message to a Java program which is reading in the document to sort the elements alphabetically. Processing instructions take the following format:

```
<?application_name message for application ?>
```

Processing Instructions may be found nearly anywhere in the XML document, including in the prolog or after the "root" element.

The Prolog

The prolog and each of its components is optional in well-formed XML documents. The prolog may contain:

- XML Declaration
- Document Type Declaration
- Document Type Definition (DTD)

XML Declaration

While the XML Declaration is not required in well-formed XML documents, its inclusion is strongly recommended. The XML Declaration identifies the document as being an XML document as well as the version of XML which it adheres to (currently 1.0). Optionally, the character set encoding and/or whether the document has external

components may also be included. The XML Declaration occupies the very first item of the document -- spaces may not even precede it. The XML declaration takes the following format:

```
<?xml version="1.0" encoding="encoding of document"
standalone="yes or no" ?>
```

Document Type Declaration

The document type declaration is used to explicitly state that the XML document adheres to a structure defined in the noted DTD or alternative schema. The DTD is either included within the document type declaration, or the location to an external DTD is given. The document type declaration including the DTD takes the following format:

```
<!DOCTYPE rootElementName [ DTD goes here ] >
```

A document type declaration giving the location to an external DTD takes the following format:

```
<!DOCTYPE rootElementName SYSTEM "URL to DTD goes here" >
```

Document Type Definition (DTD)

The document type definition, or DTD, gives the definition of the structure by describing allowable components and their relationship to other components. XML documents which adhere to their DTD may be "valid". Documents which do not adhere to their DTD are "invalid" and may be rejected by the processing application.

More precisely, the DTD allows the author to formally define the document's schema through a definition of the document's entities and elements, the elements'

attributes and contents, and the elements' relationships to other elements. A DTD may either reside inside of the document's document type declaration or reside external to the document proper. In addition, several external DTDs may be merged to form a single DTD for the document in question. This ability gives power and flexibility to the definition of the document's structure.

The XML document DTD syntax commonly used is that of the Extended Backus-Naur Form (EBNF). EBNF DTDs are also used by SGML and are a matured schema for use with text documents. However, EBNF is not an XML based notation and has numerous limitations when using XML for other non-traditional applications. Therefore, alternate XML schema languages are in development which may offer additional functionality from the classical DTD. However, at the present, the classic EBNF based DTD is the only "standard" option.

In EBNF syntax, the following DTD content model operators are available for more precise specification of a component's content and structure (Boumphrey et al., 1998):

'	strict ordering
|	selection (or)
+	repetition (minimum 1)
*	repetition (minimum 0)
?	optional
()	grouping

Element Declarations

Element declarations in a DTD take the following format:

```
<!ELEMENT ElementName ElementContents >
```

For instance:

```
<!ELEMENT myElement (childElement) >
```

Attribute List Declarations

Attribute list declarations take the following format:

```
<!ATTLIST ElementNameThatAttributeBelongsTo
AttributeName(s) CorrespondingAttribute'sDataTypeOrPossibleValues
PossibleDefaultValues >
```

For instance:

```
<!ATTLIST myElement visible (yes|no) "yes" >
```

Entity Declarations

Entity declarations for use as entity references take the following format:

```
<!ENTITY EntityName EntityValue >
```

or

```
<!ENTITY EntityName SYSTEM "URLToDTD" >
```

For instance:

```
<!ENTITY edi "Electronic Data Interchange" >
```

Parameter entity references also include a `%` as follows:

```
<!ENTITY % parament SYSTEM
"http://www.somewhere.org/myparamdef" >
```

XML Document Constraints

Well-formed XML documents were said to be syntactically correct XML documents. What exactly does that mean? The following rules capture most of the requirements: (Harold, 1998)

1. Begin the XML document with the XML declaration (not required, but strongly recommended).

2. A "root" element completely contains the document's content. All other content components must also reside in the "root" element.

3. Match start-tags with end-tags. Unlike HTML, XML end-tags are always required.

4. If an element does not contain content, the empty element tag may be used.

5. Element tags may nest, but never overlap.

6. Attribute values must be enclosed in quotes.

7. Use `&` for & and `<` for <, except in CDATA sections which by definition may contain these delimiter characters.

8. The only entity references permitted are `&` for &, `<` for <, `>` for >, `&apos;` for ', and `&quot;` for ".

Valid XML documents are essentially well-formed XML documents which include an XML declaration and document type declaration. Valid documents must also adhere to the DTD indicated in the document type declaration. Because entity references may be defined in the DTD, additional entity references may be defined and used in valid

documents.

Example XML Document

The following XML document combines most of the above mentioned components into a valid XML document:

```
<?xml version="1.0"?>
<!-- I'm a comment that just followed the XML Declaration -->

<!-- The Document Type Declaration follows next -->
<! DOCTYPE addressBook [

    <!-- DTD for addressBook -->
    <!ELEMENT addressBook (entry*)>
    <!ELEMENT entry    (name, email, address, phone, category,
note?)>
        <!ELEMENT name     (#PCDATA)>
        <!ELEMENT email    (#PCDATA)>
        <!ELEMENT address  (#PCDATA)>
        <!ELEMENT phone    (#PCDATA, emphasis*, #PCDATA)>
        <!ELEMENT emphasis (#PCDATA)>
        <!ELEMENT category EMPTY>
            <!-- attribute of category -->
            <!ATTLIST category type CDATA "unfiled">
        <!ELEMENT note     (#PCDATA|CDATA)>
    <!-- define an entity reference -->
    <!ENTITY w "waldo" >

]>

<!-- Here's the "root" element -->
<addressBook>
<!-- Here's a processing instruction -->
<?myProg sort="alpha" ?>
    <entry lastModified="1999-06-16">
        <name>Waldo</name>
        <email>waldo@nowhere.com</email>
        <!-- Did we lose him again? -->
        <address>Where <emphasis>is</emphasis> &w;?</address>
        <phone>555-555-5555</phone>
        <category type="personal"/>
        <note><![CDATA[Is his IQ < 90?]]></note>
    </entry>
</addressBook>
```

```
<!-- End of XML document -->
```

Related Technologies

In addition to the XML language itself, many complimentary technologies are in development. Current standardized technologies include:

- Namespaces
- Style Sheets
- Document Object Model

Namespaces

XML namespaces provide a means of qualifying element and attribute names with URI based namespace identifiers (W3C, 1999b). This is useful in situations where identically named elements or attributes from semantically different domains would otherwise be in conflict with each other. For instance, NAME may refer to both an element describing an organization as well as an element describing a product. Namespaces allow NAME to be used for both types of elements by qualifying each with the prefixed namespace identifier. Thus, ORGANIZATION:NAME could refer to the organization's name and PRODUCT:NAME could refer to the product's name. A namespace is defined with the following format:

```
xmlns:namespacePrefix="URIToNamespaceSchema"
```

Style Sheets

In XML, the rules for the presentation or manipulation of the data are not inherently contained within the encoded content. External means of determining how the document should be rendered must be employed. In XML, these presentation rules are usually implemented with style sheets. Style sheets contain the rendering instructions for use by the viewing application such as a web browser. In addition, some style sheets may also be used for more complex manipulation of the XML data. The W3C has provided a standard manner of associating style sheets with XML documents (W3C, 1999c).

The separation of content from style allows for multiple style sheets or processing rules to be applied to a single content source. For instance, the visual on-screen display of an XML document can be completely different from that of the printed page simply by changing the style sheet. Furthermore, another set of processing rules could be applied to this same content source which would dictate how the data is incorporated into an application program such as a database management system. However, in all these scenarios, the stored XML document remains unchanged.

Document Object Model

The Document Object Model, or DOM, is a platform and language independent application programming interface (API) which provides a means of creating, accessing, and manipulating XML documents within processing applications (W3C, 1998b). In addition to the interface methods, the DOM also represents XML documents in a tree-like data structure. The DOM is extremely useful because it allows XML to be used for

alternative, non-document-centric applications in a consistent manner regardless of the platform or programming language.

What is EDI?

Electronic Data Interchange (EDI) allows organizations to exchange business information through electronic means. At present, some 100,000 U.S. companies employ EDI (Data Interchange Standards Association [DISA], 1999). EDI can formally be defined as "the inter-organizational, computer-to-computer exchange of business documentation in a standard, machine-processable format" (Emmelhainz, 1993, p.4).

EDI has been in use for nearly 30 years, first starting in the transportation industry and quickly expanding to other areas of commerce. Today, nearly all industries make use of EDI. In fact, many companies and government agencies have made it a requirement for doing business with them.

EDI allows organizations to exchange business information more quickly, with less errors, and more cost effectively than traditional paper-based systems (Kalakota & Whinston, 1996). However, EDI should not be initiated for merely increasing the efficiency of clerical work. The resulting gains from this can be rather meager. Instead, EDI represents "a fundamental change" in the way organizations do business (O'Callaghan, Kaufmann, & Konsynski, 1992, p.47). To fully reap EDI's benefits, organizations must be willing to fully reengineer their business processes. EDI should be seen as an "enabling technology" and "change driver" to allow this overarching transformation to take place, and not as a solution to a specific problem (Colberg,

Gardner, Horan, McGinnis, McLauchlin, & So, 1995, p.3).

If properly implemented, the rewards of successful EDI initiatives are phenomenal. Without EDI, a single order may cost approximately $75-125 to fully process. With EDI, that cost could drop to less than $1.00 (Colberg, Gardner, Horan, McGinnis, McLauchlin, & So, 1995). An important aspect of EDI initiatives is the emphasis placed on the relationships with other organizations. In fact, "an appropriate use of EDI ... [is the] ... transition to a leaner production process through new inter-organizational relationships" (Gottardi & Bolisani, 1996). For instance, the automobile industry employs EDI in its just-in-time inventory system to allow them to order and receive parts just minutes before they are needed on the assembly line. This reduces the automobile manufacturers need for costly stock piles of inventory and warehouses (Ritter, 1998). Other examples of EDI's benefits abound.

The elevated importance of inter-organizational relationships in EDI initiatives does come at a price. EDI relies on strong technical capabilities from both the internal organization *and* the trading partner. In other words, if one trading partner has not implemented EDI to the same degree as the other, then overall efficiency is ultimately effected. It is this focus on the trading partner that necessitates the diffusion of successful EDI initiatives to all clients and vendors of an organization. Thus, it is in the EDI community's best interest to expand EDI adoption and effectiveness.

Components of EDI Systems

In order for two humans to communicate, they must find some common language.

Even if the communication is irregular and garbled, humans are often able to infer the intended meaning. Computers, however, are usually not able to do this.

EDI provides a means for two autonomous computers to exchange business data. The infrastructure of EDI consists of three main components. First, a predefined, standard message format must exist so that each computer is able to process the exchanged document according to some predefined algorithm. Second, the sending computer must employ translation software in order to extract the necessary information to be exchanged from existing data stores and format it into the standard message format. (The computer on the receiving end needs to do the reverse of this process.) Finally, some type of communications link must exist to allow the computers to exchange the messages.

Standard Message Format

EDI is the exchange of structured business documents. Unstructured documents such as informal person-to-person email messages are not part of the EDI framework. Furthermore, non-repetitive transactions are also beyond EDI's scope. EDI focuses on automating relatively static business relations by creating a reusable electronic business message.

The structure of the interchange message can be created from scratch by the two trading partners. However, this may necessitate creating a new message format for each trading partner, which quickly grows unmanageable. Therefore, standards bodies have produced flexible standard message interchange formats. The message formats are designed to allow organizations to exchange messages with multiple, cross-industry

trading partners by providing a common base message format and also optional extensions for meeting specialized requirements.

The dominant EDI standard in North America is maintained by the ANSI ASC X12 committee. The United Nations' sanctioned EDI standard, UN/EDIFACT, enjoys high use in international markets. UN/EDIFACT is very similar to ANSI ASC X12. In fact, ANSI has voted to eventually make X12 compliant with EDIFACT. This paper will focus on the X12 standard.

ANSI ASC X12 Message Format

In X12 terminology, a business document is called a transaction set. There are hundreds of transaction sets in the current X12 standard. Each transaction set is given an 3 digit ID designation. For instance, a request for a quotation is transaction set 840, a purchase order is transaction set 850, an invoice is transaction set 810, etc.

Transaction sets are made up of data segments. Data segments correspond to lines in a traditional business document. In turn, data segments are made up of data elements. Data elements correspond to the separate pieces of information contained on a line.

Please refer to the following hypothetical paper-based purchase order (Emmelhainz, 1993, p.62):

```
P.O. Number 4001
P.O. Date December 31, 1992
Buyer: Allen Manufacturing
123 North Street
Largetown, NY 11111
Vendor: Baker Supplies
P.O. Box 989
Somewhere, NY 10009
Ship to: Plant 1
456 West Ave
Smallsville, NY 10006
```

```
5 cases part number BC436 @$12.50/cs
Number of line items
```

The above document could be considered a transaction set. An individual line, such as "Smallsville, NY 10006", could be considered a data segment. The city component of this data segment, "Smallsville", could be considered a data element.

Each transaction set has mandatory, optional, and conditional data segments. Mandatory segments are required to be present. Optional segments are just that. This provides flexibility by allowing trading partners to use only those optional segments which they require. Conditional segments are segments which must be present if a defined condition is met such as the presence of an optional segment. Some segments may be repeated a specified number of times.

The hypothetical paper-based purchase order presented earlier may translate to the following X12 850 purchase order with "*" used as the data element delimiter and "^" used as the segment terminator (Emmelhainz, 1993, p.62):

```
ST*850*001^
BEG*00*NE*4001**921231^
N1*BT*Allen Manufacturing^
N3*123 North Street^
N4*Largetown*NY*11111^
N1*VN*Baker Supplies^
N3*P.O. Box 989^
N4*Somewhere*NY*10009^
N1*ST*Plant I^
N3*456 West Ave^
N4*Smallsville*NY*10006^
P01*1*5*CA*12.50**VP*BC436^
CTT*1^
SE*14*0001^
```

Multiple transaction sets can be combined into a single EDI interchange. The first and last lines of the example X12 850 purchase order are the Transaction Set Header and

Transaction Set Trailer respectively. Different types of transaction sets can also be sent in a single interchange through the use of functional group. All transaction sets of a common type are placed in one functional group. Each functional group receives a header and trailer line in the EDI message. Finally, the whole EDI interchange message receives a header and trailer indicating the sender and recipient of the message.

Thus, an ANSI ASC X12 EDI message is highly structured and flexible in design. The outer most "envelope" indicates the sender and receiver. Inner envelopes indicate the functional group of transaction sets contained therein. Multiple transaction sets of a like type may be found inside a functional group. Each transaction set contains segments, which may be mandatory, optional, or conditional for that transaction set. A segment contains data elements which may be mandatory, optional, or conditional.

Negotiating the X12 Message Format

When two trading partners agree to exchange X12 EDI messages, they must agree which transaction sets they will be exchanging, as well as the version of X12 they will be implementing. Next, they must determine the structure of those transaction sets. X12 provides transaction set tables to allow trading partners to determine which segments are available for a particular transaction set. The table indicates the segments' ID, order, inclusion requirements, and the number of times a specific segment can be repeated. Trading partners must agree which segments they will be using for each transaction set.

After determining the transaction set and which segments they will be using, trading partners must now determine the structure of each segment and which data elements for that segment they will support. Trading partners can reference a segment

diagram key for each segment. The segment diagram keys contain the segment ID, data element delimiter, element diagrams, segment terminator, and additional notes. Data elements for a particular segment may be mandatory, optional, or conditional.

The element diagrams of the segment diagram keys indicate the order and inclusion requirements of data elements of a segment. Element diagrams are sequentially ordered in the segment diagram key and provide the data element reference designator, title, data dictionary reference number, requirement designator, type, and length. The reference number is the made up of the segment ID plus a 2 digit number. The data element dictionary may be referenced for further information about a data element.

After the preceding message components have been determined, trading partners must also map their institution specific information between the two. For instance, internal part numbers, product descriptions, etc., must be correlated.

Translation Software

In essence, EDI translation software performs the role of converting the standard EDI interchange format into some format useful to the local organization. In a non-integrated system, this software may simply convert the electronic EDI interchange into a human readable printed document. On the other hand, organizations with highly integrated EDI implementations require specialized software to perform the following steps in order to be able to create and send an EDI message (Emmelhainz, 1993):

- Mapping
- Extraction (or conversion)

- Generation (outbound) or Interpretation (inbound)
- Communication

Mapping

The first step of mapping involves identifying where the relevant information to be interchanged resides in the local organization's information system. This step requires knowledge of both the EDI standards as well as the local information system and requires in-house programming. Mapping must take place before any other components of EDI software are installed (Emmelhainz, 1993).

Extraction

The extraction step gathers the previously mapped data and puts it in a temporary format, typically a flat file. The exact format of the temporary structure is usually dictated by the translator software. Since local data structures and syntaxes often vary, this step usually requires in-house programming.

Generation

Generation involves converting the temporary flat file into the EDI format. Commercial EDI translation software is typically employed at this stage. EDI translators are usually aware of several different message standards and can be easily updated as the standards change. EDI translation software also perform checks on the message to ensure it is properly formatted.

Communication

Once the EDI interchange has been generated, it is ready to be sent to the trading partner. Communication software makes the necessary connections and transports the EDI message.

Communications Network

EDI messages are typically communicated in the following manners:

- Direct link
- Value Added Network
- the Internet

Direct Link

Direct links typically employ leased lines or use dial-up modems to provide a point-to-point connection between two trading partners. Trading partners must ensure that they have compatible hardware and are using the same communications protocol. Direct links are most useful when time sensitive real-time exchanges are required. For instance, when EDI is employed in just-in-time inventory systems, it is crucial that EDI messages are instantly received. Direct links are also useful in situations where an organization has only one other trading partner. However, as the number of trading partners increases, it may grow too unmanageable or expensive to employ direct links.

Value Added Network

Using the services of a Value Added Network (VAN) is the dominant method for transporting EDI messages. A VAN provides an organization with one entry point for interchange. This reduces the need for additional network equipment and configuration as is found in situations with multiple direct links. VANs typically provide an electronic mailbox for organizations to send, store, and receive EDI messages. Ideally, all trading partners of an organization would also use the same VAN. VANs may also provide additional services such as interconnections with other VANs, increased audit tracking, security, error checking, translation, and media conversion. As can be expected, VANs often charge a premium for their services and may be beyond the financial scope of smaller organizations.

Internet

Because of its ubiquity and low cost, companies are exploring the use of the Internet for EDI message transport. In such implementations, EDI messages are usually sent as S/MIME messages to an Internet email account. Perhaps the biggest inhibitors to the use of the Internet for message transport are concerns over reliability and security. Organizations also lose the additional services provided by a VAN.

Barriers to EDI Adoption

In some respects, EDI has failed to live up to its potential. It is not as essential to

business as is the telephone or fax machine. In reality, it is really not all that widespread -- nearly 80% of EDI transactions come from the top 20% of the highest volume submitters (Cupito, 1998, p.32). In short, EDI is not ubiquitous. Thus, EDI actually limits the number of trading partners an organization may have because of the lack of alternatives and the need for well defined agreements (Gottardi & Bolisani, 1996, p.381).

EDI barriers are those factors which effect its adoption or use. Research has shown that adoption barriers more or less relate to the following:

- Unsatisfactory message standards
- Systems incompatibility
- High costs

Unsatisfactory message standards

In a nutshell, EDI is too rigid. It is not capable of describing the "exception conditions" which frequently occur (Crum, Premkumar, & Ramamurthy, 1996, p.50). EDI attempts to provide 'universal' document structures which are adequate for all potential uses. However, this approach is now seen as "quite impossible, even conceptually" (Gottardi & Bolisani, 1996, p.379).

As a result, X12 is not enough by itself. The implementer must also incorporate industry specific "implementation guidelines" and trading-partner-specific "implementation conventions" (Lehmann, 1996, p.28). Thus, even if one follows the EDI 'standards', it does not mean that messages can be exchanged with just anyone. This

specificity restricts EDI to "well bounded communities" (Gottardi & Bolisani, 1996, p.387). Moreover, it also brings "into question the success of the whole EDI standardisation process" (Steel 1996, p.14).

As the EDI standards continue to grow, they continue to be "held back by inflexibility and complexity for users" (Lehmann, 1996, p.41). Instead, the standards must be marked by flexibility and simplicity. EDI must provide a simple infrastructure for creating truly flexible document structures which can be exchanged with nearly anyone.

Systems Incompatibility

For maximum returns, EDI implementations must be highly integrated with the organization's internal systems. In practice, however, over 70% of all EDI actually takes place with computers having no role in the transaction other than in transmission (Lehmann, 1996, p.28). In this scenario, EDI interchanges are simply emailed or faxed EDI forms, written and read by human beings. Certainly better alternatives exist.

Unfortunately, few packaged information systems come EDI-enabled. Software producers cite the extreme complexity of EDI standards and the required staff training costs as being too substantial to justify the cost of EDI-enabling their products (Steel, 1996). Perhaps part of the problem is that "current methods of standardisation of the structure of data being interchanged across a machine-to-machine interface totally ignore the way in which applications and programs are designed and operated" (Steel, 1996, p.15).

Thus, nearly every implementer of EDI is faced with the problem of how to adapt their internal systems for use with EDI (Chen, 1998). The solution often involves extensive reprogramming. This has been shown to be an empirically significant barrier to EDI adoption (Crum, Premkumar, & Ramamurthy, 1996; O'Callahan, Kaufmann, & Konsynski, 1992)

EDI needs to become less complex, designed with the requirements of processing applications in mind, and utilize more generally available programmer skills in order for software vendors to begin incorporating EDI into their products. This would help EDI reach a critical mass of availability where it truly could become required for business in the contemporary market. Moreover, as the skills necessary for systems integration tasks become less specialized, the task of fully integrated systems would be less formidable.

High Costs

Perhaps the biggest impediment to EDI adoption is the high costs associated with EDI implementation (Crum, Premkumar, & Ramamurthy, 1996; Cupito, 1998). In many situations, EDI is simply not cost effective. This is particularly true for smaller organizations with limited volume.

EDI's costs are predominately associated with EDI enabling software, maintenance from standards evolutions, and telecommunications charges (Millman, 1998). In addition to the previously mentioned changes in message standards and systems compatibility, increased used of low cost, ubiquitous transport mechanisms such as the Internet could lessen some of the costs while increasing the number of potential trading

partners.

Using XML for EDI

EDI, at its core, is a set of standard, structured electronic documents which are machine processable. As has been shown, XML also can be used to create machine processable electronic documents which adhere to predefined structures. However, XML also offers additional benefits: XML is relatively simple, quite flexible, Internet aware, easy to program, and seemingly on the road to ubiquity. This paper will now explore how XML can be used in each of the components of an EDI system and/or XML's effect on the component.

Standard Message Format

On a most simplistic level, XML can be used for the encoding of EDI messages. For instance, the example purchase order presented in the EDI section of this paper could be represented in XML as follows:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE purchase-order SYSTEM
  "http://www.company.com/dtd/purchase-order" >
<purchase-order>
  <po-number>4001</po-number>
  <po-date>1992-12-31</po-date>
  <buyer>
    <name>Allen Manufacturing</name>
    <billing-address>
      <street>123 North Street</street>
      <city>Largetown</city>
      <state>NY</state>
```

```
        <zipcode>11111</zipcode>
      </billing-address>
      <shipping-address>
        <building>Plant 1</building>
        <street>456 West Ave</street>
        <city>Smallsville</city>
        <state>NY</state>
        <zipcode>10006</zipcode>
      </shipping-address>
    </buyer>
    <vendor>
      <name>Baker Supplies</name>
      <address>
        <po-box>989</po-box>
        <city>Somewhere</city>
        <state>NY</state>
        <zipcode>10009</zipcode>
      </address>
    </vendor>
    <item>
      <line-item>
        <quanity>5</quantity>
        <unit>case</unit>
        <unit-price currency="USD">12.50</unit-price>
        <part-number>BC436</part-number>
      </line-item>
    </item>
  </purchase-order>
```

XML provides verbose, machine *and* human readable data structures. Even without knowledge of XML or EDI, a human can more or less determine the structure and content of the above document. This is a positive step in reducing the complexity of EDI.

XML documents have the capacity to be self describing through the use of associated DTDs. This is an improvement over EDI interchanges which may only be interpreted through reference to external, and more than likely, non-machine processable implementation guidelines.

However, the real power of XML for EDI lies in XML's flexibility for creating and extending document structures through the use of DTDs. An XML DTD can merge

several pre-existing and standardized DTD fragments to make a structure optimized for the particular trading partners needs. Thus, the design of EDI message structures can be rather à la carte in nature, based entirely on unique combinations of standard DTD fragments.

For instance, the example XML-based purchase order DTD may be as follows:

```
<!-- DTD for purchase-order -->

<!ELEMENT purchase-order (po-number, po-date, buyer, vendor,
item)>

<!-- DTD fragments defined by standards body -->
<!ENTITY % pon SYSTEM "http://www.standards.org/dtd/po-number">
<!ENTITY % pod SYSTEM "http://www.standards.org/dtd/po-date">
<!ENTITY % buyer-contents SYSTEM
"http://www.standards.org/dtd/buyer">
<!ENTITY % vendor-contents SYSTEM
"http://www.standards.org/dtd/vendor">
<!ENTITY % item-contents SYSTEM
"http://www.standards.org/dtd/item">

<!ELEMENT po-number %pon>
<!ELEMENT po-date %pod>
<!ELEMENT buyer %buyer-contents>
<!ELEMENT vendor %vendor-contents>
<!ELEMENT item %item-contents>

<!-- end of DTD for purchase-order -->
```

In this scenario, the standards organization controls the structure of the elements such as `buyer`. However, the trading partners could further specify/override the elements as needed. This provides both increased flexibility and greatly simplifies the process of writing processing applications.

Of course, the real danger of XML use for EDI lies in the great ease of defining new tags. In the rush to implement XML, organizations may neglect to the standards bodies and create all of their own tags. This has the potential of creating a virtual Tower

of Babel of competing 'standards'. Thus, standards bodies must be quick to provide organizations with accessible and standard DTD fragments (XML/EDI Group, 1998).

Translation Software

With XML-based EDI, the function of mapping would still be required and perhaps unchanged. However, the extraction and generation of EDI interchanges would be greatly altered and simplified. This should positively affect the costs of EDI systems integration since the extraction phase of EDI translation can cost over 15 to 20 times the cost of the EDI translator software (Lehmann, 1996, p.32).

As previously noted, the lack of EDI compatible software often requires expensive custom programming for effective EDI implementations. While EDI compatible software is relatively scarce, vendors are scrambling to implement XML interfaces into their products. This promises to create a large body of developers who are skilled in XML based technologies. Thus, developers who work on XML-based EDI initiatives involving data extraction would be more obtainable then the current small, specialized EDI community. In addition, vendors may be more willing to incorporate EDI into their applications if they see EDI as being less esoteric. Thus, the availability of EDI-enabled products would be increased.

XML's ability to associate multiple style sheets and processing applications to a single document has many benefits which extend beyond the traditional computer-to-computer use of EDI. For instance, XML documents may have also have style sheets

associated with them which are optimized for functions such as printing. This would increase the utility of an EDI interchange without the need for specialized software. Moreover, websites could provide human interfaces to XML based EDI forms, further increasing the exposure and usefulness of the EDI system.

Communications Network

A major inhibitor to the adoption of EDI has been the use of VANs for transport due to their high costs. While XML documents may themselves be transported by nearly any means including VANs, XML is Internet savvy by design. Thus, XML based EDI implementations may find themselves heavily Internet based. The low cost and ubiquity of the Internet promises to bring EDI use into the realistic grasp of many smaller organizations and the international markets.

However, VANs may still be appealing because they provide many services in addition to simple message transport. By focusing on the value-added aspects of their services, VANs may be able to transform themselves into Internet-based Value Added Virtual Private Networks, acting more as gateways and less as network providers. This could extend their reach beyond to organizations who previously did not utilize VANs.

Conclusions

This paper examined XML and EDI to see how XML could be used for EDI. It has been shown that XML can indeed be used for EDI. More importantly, XML does appear to address the barriers to EDI adoption. Using XML for EDI promises to provide less complex and more flexible message standards, more easily integrated EDI systems, and lower costs. Perhaps XML can breathe new life into EDI.

XML offers particular appeal to the smaller organization wishing to implement EDI. Traditional forms of EDI have often been beyond their technical and financial reach. However, XML based EDI promises to lower costs and utilize more generic skills. Perhaps XML can lead to EDI's ubiquity.

XML also offers the potential for creating synergy between the developers working on business-to-consumer and business-to-business aspects of electronic commerce. The years of electronic commerce expertise found in the EDI camps could then be leveraged into to the relatively new web retailing market. This has obvious impact on all businesses which offer electronic storefronts.

However, XML does have some problems which need to be addressed. Standards organizations need to quickly provide DTDs before industry creates a incompatible web of their own "standards". It would be ironic if XML's great advantage of extensibility were to lead to its demise.

In addition, the limited nature of the DTD in matters such as the lack of data type specification (numeric, date, etc.) seriously undermines its usefulness. However, the alternative schemas currently under development do address this very problem. Moreover, external entity references provide a means of directly incorporating non-XML data into the XML document.

In the end, the ball is in the standards organizations court. They must provide a straightforward XML-based solution which is easy to implement. After all, the simpler, "good enough" solution will beat a highly complex solution, even if the later is "better".

References

Boumphrey, F., Direnzo, O., Duckett, J., Graf, J., Houle, P., Hollander, D., Jenkins, T., Jones, P., Kingsley-Hughes, A., Kingsley-Hughes, K., McQueen, C., & Mohr, S. (1998). *XML applications*. Olton, Birmingham, U.K.: Wrox Press.

Business-to-business e-commerce. (1999, June). *Mobile computing & communications*, p. 30.

Chen, J. (1998). The impact of Electronic Data Interchange (EDI) on SMEs: Summary of eight British case studies. *Journal of small business management*, *36*(4), pp.68-72.

Crum, M. R., Premkumar, G., & Ramamurthy, K. (1996). An assessment of motor carrier adoption, use, and satisfaction with EDI. *Transportation journal*, *35*(4), pp.44-57.

Cupito, M. (1998). Obstacles to ubiquity: EDI's slow acceptance. *Health management technology*, *19*(4), pp.30-34.

Colberg, T. P., Gardner, N. W., Horan, K. J., McGinnis, D. M., McLauchlin, P. W., & So, Y. (1995). *The Price Waterhouse EDI handbook*. New York: Wiley.

Data Interchange Standards Association [DISA]. (no date). *Realize a profit-filled tomorrow* [Online]. Available: http://www.disa.org/x12/membership/whatedit.htm [July 6, 1999].

Emmelhainz, M. A. (1993). *EDI: A total management guide* (2nd ed.). New York: Van

Nostrand Reinhold.

Gottardi, G., & Bolisani, E. (1996). A critical perspective on information technology management: The case of electronic data interchange. *International journal of technology management*, *12*(4), pp.369-390.

Harold, E. R. (1998). *XML: Extensible markup language*. Foster City, CA: IDG Books Worldwide.

Kalakota, R., & Whinston, A. B. (1996). *Electronic commerce: A manager's guide*. Reading, MA: Addison Wesley.

Lehmann, F. (1996). Machine-negotiated, ontology-based EDI (Electronic Data Interchange). In N. R. Adam & Y. Yesha (Eds.) *Electronic commerce: Current research issues and applications* (pp. 14-45). New York: Springer.

Millman, H. (1998, April 6). A brief history of EDI. *InfoWorld* [Online]. Available: http://archive.infoworld.com/cgi-bin/displayTC.pl?/980406sb5-edi.htm [July 6, 1999].

O'Callaghan, R., Kaufmann, P. J., & Konsynski, B. R. (1992). Adoption correlates and share effects of electronic data interchange systems in marketing channels. *Journal of marketing*, *56*, pp. 45-56.

Ritter, D. (1998, December). What to do about EDI. *Intelligent Enterprise*, pp. 74-77.

Shih, C., & Terhune, A. (1998). *A harbinger of things to come?* [Online]. Available: http://help.unc.edu/gartner/research/ras/72700/72783/72783.html [July 6, 1999].

Steel, K. (1996). The standardisation of flexible EDI messages. In N.R. Adam & Y. Yesha (Eds.) *Electronic commerce: Current research issues and applications* (pp.

13-26). New York: Springer.

XML/EDI Group. (1998). *Guidelines for using XML for electronic data interchange* [Online]. Available: http://www.geocities.com/WallStreet/Floor/5815/guide.html [July 6, 1999].

World Wide Web Consortium [W3C]. (1998a). *Extensible markup language (XML) 1.0* [Online]. Available: http://www.w3.org/TR/REC-xml [July 6, 1999].

W3C. (1998b). *Document object model (DOM) level 1 specification* [Online]. Available: http://www.w3.org/TR/REC-DOM-Level-1/ [July 6, 1999].

W3C. (1999a). *W3C Extensible markup language (XML) activity* [Online]. Available: http://www.w3.org/XML/Activity.html [July 6, 1999].

W3C. (1999b). *Namespaces in XML* [Online]. Available: http://www.w3.org/TR/REC-xml-names [July 6, 1999].

W3C. (1999c). *Associating style sheets with XML documents* [Online]. Available: http://www.w3.org/TR/xml-stylesheet [July 6, 1999].