DESIGNING AND IMPLEMENTING THE DTD INFERENCE ENGINE FOR THE I-WIZ PROJECT

By

HONGYU GUO

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2000

To my wife Yanping, and my daughter Alicia, who was born during this thesis work.

ACKNOWLEDGMENTS

I owe my success in the research work and my career in computer science to my research advisor, Dr. Joachim Hammer. Dr. Hammer introduced me to this interesting area of database integration using XML and related technologies. I benefited from his vision, his broad and deep knowledge and his rich experiences in the database area. I am also grateful to Dr. Douglas Dankel and Dr. Abdelsalam Helal, who gave me good supervision as members of my supervisory committee. My special thanks go to Dr. Dankel. As a department graduate advisor, he has given me invaluable advice on both curricula and careers.

I am also thankful to the group members of the I-Wiz project, especially Charnyote Pluempitiwiriyawej and Amit Shah. I benefited from the group meetings as well as individual discussions. My work would not have been possible without the strong support and sacrifice of my wife Yanping and the cooperation of my daughter Alicia. While Yanping has her own graduate study, she handles most of the care for the baby as well as the household chores. I also thank my daughter Alicia, who was born in the middle of this thesis, for being such an easy-going and sweet girl. She always smiles and hardly cries. When I need to get the work done, I just talk politics to her and she goes to sleep.

TABLE OF CONTENTS

page

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

DESIGNING AND IMPLEMENTING A DTD INFERENCE ENGINE
FOR THE I-WIZ PROJECT

By

Hongyu Guo

August 2000

Chairman: Joachim Hammer
Major Department: Computer and Information Systems and Engineering

DTD (Document Type Definition) inference for XML is an active research area. The implementation of the DTD Inference Engine (DIE) is also a part of the I-Wiz project, an on-going XML-based database integration project in the University of Florida. This thesis addresses the research and implementation issues underlying DTD inference. We make theoretical investigations into DTD inference in the context of context-free languages and present our design and implementation of a DTD Inference Engine.

Our theoretical study clarifies some concepts that have been used intuitively and vaguely without definition in the research literature. We define the essential concepts such as a kernel derivation tree (KDT), sound, tight and closure DTD. We introduce two theorems stating the relationship between multiple kernel derivation trees from a single grammar and multiple grammars to a single derivation tree. We conclude that a finite language has a finite number of KDTs and an infinite language has an infinite number of

KDTs. We reveal the possibility of nonexistence of closure DTDs for some source XML documents.

We state our choice of the rules for DTD inference and we give our rationale for the choice of the rules. We design a new unique architecture for the DTD Inference Engine and a three-dimensional linked list data structure for representing the inferred DTD internally. We give algorithms and complexity analysis of the engine. Our DIE has three unique features not found anywhere else: a factorization reduction, the ability to handle multiple documents mechanism, and the incremental maintenance. Finally, we describe several possibilities for extending this work.

CHAPTER 1
INTRODUCTION

The need for integrating heterogeneous data sources, especially web data sources has been recognized as an important IT problem for this century. Some integration systems for Internet sources have already been developed in the past few years, like one-stop bookstores and one-look dictionaries. At an on-line, one-stop bookstore one can search for books sold by multiple on-line bookstores and compare prices. Similarly, if someone wants to lookup a word, especially if it is not a very common word, he/she can consult a one-look dictionary, for example, since the word may be hard to find in any particular dictionary. While he/she searches a word or phrase in the one-look dictionary, the dictionary returns definitions from one or more on-line dictionaries for reference.

Different approaches and architectures for integrating heterogeneous information from different sources have been proposed [1-7]. In addition to the more traditional research on integrating structured data, we have seen an increase in the study of semi-structured data [8,9]. The LORE project at Stanford University [10,11] is an example. With the advent of XML [12], semi-structured data has found a new representation. Although XML was intended to provide a platform independent format for data exchange, it is also a natural representation to describe semi-structured data. Since the first proposal for XML specifications by the World Wide Web Consortium in 1998 [12], XML has gained popularity and it will be the dominant Internet data format for the future, especially for e-commerce applications. For example, LORE has already migrated

to XML [13-15] shortly after the proposed standard of XML by the World Wide Web Consortium.

A characteristic that distinguishes XML from other semi-structured data models is the notion of a Document Type Definition (DTD) that may optionally accompany an XML document. A document's DTD serves the role of a schema specifying the internal structure of the document. As we will show later, a DTD specifies, for every element, the regular expression pattern that subelement sequences of the element need to conform to. DTDs are critical to realizing the promise of XML as the data representation format that enables free interchange of electronic data (EDI) and integration of related information (e.g. news, products, services) from disparate data sources. This is because in the absence of a DTDs, tagged documents have little or no meaning. However, once the major software vendors and corporations agree on domain-specific standards for DTD formats, it would become possible for inter-operating applications to extract, interpret and analyze the contents of a document based on the DTD to which it conforms.

In addition to enabling free exchange of electronic documents, DTDs also provide the basic mechanism for defining the structure of the underlying XML data. As a consequence, DTDs play a crucial role in the efficient storage of XML data as well as the formulation, optimization and processing of queries over a collection of XML documents. For instance J. Shanmugasundaram et al. describe an approach of querying XML documents using standard commercial relational database systems [16]. A. Deutsch et al. [17] also explore ways to use relational database management systems to store and manage semi-structured data, specifically on XML data. They store frequently occurring portions of XML documents in the relational system, while the remainder is stored in an

overflow graph. Again, a DTD is needed to simplify the overflow mapping. M. Fernandez and D. Suciu also showed the importance of a DTD in optimizing regular path expression queries using graph schemas [18]. DTDs are also used in the DataGuide of the Lore project to devise efficient plans for queries and to speed up query evaluation in XML databases by restricting the search to only relevant portions of the data [19,20].

Despite their importance, however, DTDs are not mandatory and an XML document may not have an accompanying DTD. Several recent papers claim that only specific portions of XML database have associate DTD, while the rest is "schema-less." For example, large volumes of XML documents are automatically generated from data stored in relational databases, text files (e.g. HTML files, bibliographic files) or other semi-structured repositories and these XML documents do not have DTDs. Therefore, based on the above discussion on the virtues of the DTD, it is important to devise algorithms and tools that can infer an accurate, meaningful DTD for a given collection of XML documents (i.e., instances of a DTD).

This is not an easy task. Since the DTD syntax incorporates the full specification power of regular expressions, manually deciding such a DTD schema for even a small set of XML documents created by a user, could be a very complex process. Furthermore, as we show in this thesis, naive approaches fail to deliver meaningful and intuitive DTD descriptions of the underlying data. Both problems get worse as the size of XML documents increases.

Because of the importance of the DTDs, numerous potential applications depend on efficient, automated DTD inference tools. For example, the I-Wiz project, an on-going research project on XML-based database integration in the Database Center at the

University of Florida, utilizes our DTD Inference Engine. We give a more in-depth description of the I-Wiz project [21] in Chapter 2.

In this thesis, we design an efficient data structure to represent a DTD and describe the architecture of our DTD Inference Engine, a new system for inferring an accurate, meaningful DTD schema for a repository of XML documents. A naïve and straightforward solution to the DTD inference problem would be to infer as the DTD for an element, a concatenation of all the sections of sequences exactly as seen in the document. However, as we will outline in this thesis, the DTDs generated by this approach tend to be redundant and unintuitive. In fact, we discover that accurate and meaningful DTDs are also intuitive, and tend to generalize. That is to say, "good" DTDs are typically regular expressions describing subelement sequences that may not actually occur in the input XML document per se. It is important to note that this is always the case for DTD regular expressions that correspond to infinite languages, e.g. DTDs containing one or more Kleene stars [22]. In practice there are numerous such candidate DTDs that generalize the subelement sequences in the input, and choosing the best DTD that is best is a nontrivial task.

In the inference algorithm developed in this research, we propose techniques to generalize DTDs that effectively capture the structure of the input sequence. There are three novel features of our new approach: factorization, multiple document handling, and incremental maintenance of inferred DTDs. The approach as well as the novelties and contributions are described in this thesis.

The remainder of this thesis is organized as follows: In Chapter 2, we briefly describe the I-Wiz project, which provides the framework for the DTD Inference Engine

proposed in this research. In Chapter 3, we discuss the related research including the Lore project and other DTD generators, as well as theoretical studies on DTD inference including context free grammar (CFG). In Chapter 4, we describe the XML specification for the syntax of DTDs. We list the complete specification in EBNF form in the appendix. In Chapter 5 we give our theoretical results about DTD inference including definitions and motivation for Kernel Derivation Tree, Sound, Tight and Closure DTD. We investigate the relationship between the DTD as grammar and the XML source documents as the derivation trees by giving two theorems. Chapter 6 is the description of the design and implementation of the DTD Inference Engine. In Chapter 7 we discuss the incremental maintenance issue. Chapter 8 concludes the thesis and outlines possible future work.

CHAPTER 2
THE I-WIZ PROJECT

This thesis work is part of the I-Wiz project, investigating the integration of heterogeneous data sources using XML as a common representation. In this chapter we give an overview of the I-Wiz project, which provides the framework and underlying infrastructure for our DTD Inference Engine. The I-Wiz project and its team member are part of the database center at the University of Florida.

The I-Wiz project provides a solution handling increasing amount of interesting data stored in heterogeneous, other web-based sources. It focuses on sources containing semi-structured data. Its overall goal is to provide integrated access to heterogeneous data through one common interface and user-definable views of the integrated data. In addition, it provides a data warehouse to cache frequently accessed data for faster retrieval. It simplifies the access to heterogeneous information in the following three ways: (i) It helps users describe the desired information in a format that is suitable to their needs by defining a view of the desired information. This view may be as simple as a list of concepts or as complex as a schema containing entities, attributes, and relationships. (ii) It resolves semantic heterogeneity by automatically restructuring and transforming the relevant source data into a unified domain model. (iii) It Supports the querying of source data through user-defined views and the transferring of the selected data into the view definition with all inconsistencies resolved. I-Wiz makes information access ubiquitous

by using the WWW as an interface to I-Wiz. Further, the intended representation of data and schema is in the form of XML documents and DTDs.

Figure 1 shows the overall architecture of the I-Wiz project.

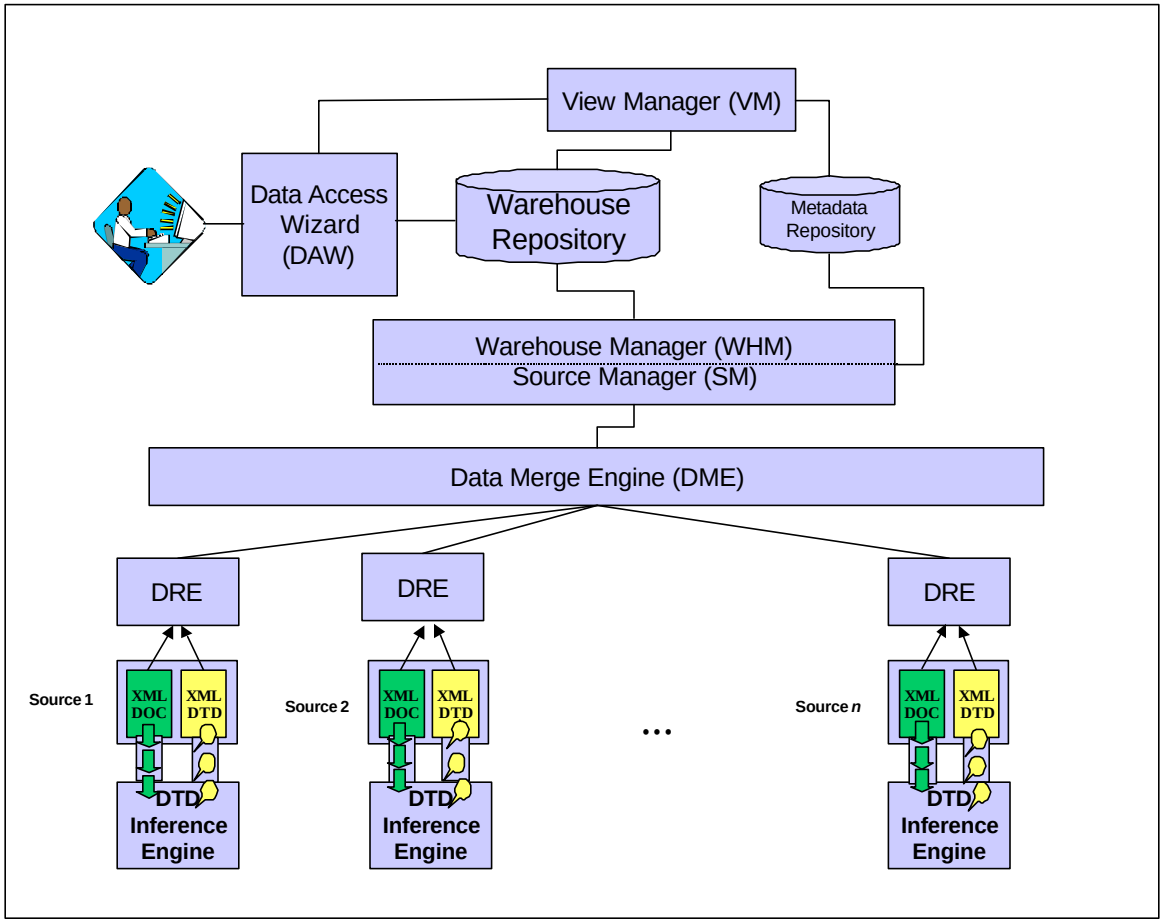

Figure 1. Overview of the I-Wiz Architecture

At the core is the Data Warehouse Repository. The Data Warehouse stores the data for the various target schemas (one per application domain), the ontology for describing the terms in each schema and the data for the user views (if they are materialized). The Metadata Repository stores the metadata for the views and the target

schemas used by the Data Warehouse Repository (one for each application domain). The View Manager, shown on top of the Data Warehouse Repository, creates and manages the user views.

The Data Access Wizard allows users to view data in an easy-to-understand and easy-to-explore fashion using a Web-based graphical interface. Rather than writing queries using a formal query language, the user can view and manipulate the data in a query language independent, drag-and-drop fashion. Data manipulation includes the browsing of a global (target) schema for each supported application domain, the browsing of the corresponding ontology defining the concepts and terms used in the schema, the definition of views which are based on the global schema, as well as the browsing and querying of user views.

The Source Manager and the Warehouse Manager interact closely with each other. The task of the source manager is to maintain a catalog of sources including a description of the data that is stored in each source, query capabilities, schema information and other information that is useful and necessary for locating sources that can contribute data to the target schema. It uses the metadata repository to make source info persistent.

The task of the Warehouse Manager is to maintain the contents of each global schema stored in the warehouse. It is thereby responsible for incorporating the merged XML data into the corresponding global schema in the warehouse. This is done by either overriding the existing contents, appending the new XML to the existing data in the warehouse or merging the two.

At a lower level, the Data Merge Engine is responsible for fusing the data in different sources into one representation described by the global target schema before

they are able to be stored in the Data Warehouse. An important part of this fusion step is the reconciliation of conflicts that typically exist in overlapping data coming from multiple sources.

Because of the flexibility of XML source documents, sources may have very different structures. Before merging, we need to restructure the result data that may be returned from the multiple sources, according to a pre-defined global ontology. The Data Restructuring Engine [21] transforms the structure and syntax of the XML data representing the answer into semantically equivalent XML documents that conform to the structure outlined in the target schema in the data warehouse.

The Data Restructuring Engine needs both the global DTD and the DTD of each source document. In those cases where the sources do not have DTDs, the DTD Inference Engine is needed. The DTD Inference Engine takes an XML document or a set of XML documents representing the source, explores their internal structures, and output a sound, intuitive and most general DTD, which describes the source(s). In the next chapter, we will summarize the active research that is related to our work.

CHAPTER 3
RELATED RESEARCH

Inference in structural information sources describing the structure of data is an active research area. In this chapter, we indicate the horizontal connections of our DIE to current research. Especially we will investigate the ongoing research in three related areas: storage and management of semi-structure data, other tools and approaches to inferring DTDs, as well as theoretical work on DTD inference, including Context Free Grammars.

3.1 Storage and Management of Semi-structured Data

As the number of information sources, which are accessible electronically, is growing rapidly, many of these sources store and export unstructured data instead of the traditional structured data, or they combine the unstructured and structured data. In most cases, however, the unstructured data are not entirely devoid of structure, i.e. they are semi-structured. Data are considered semi-structured when the underlying schema are not fixed or known in advance or when the data are incomplete or irregular. Traditional databases, for example, those based on relational and object-oriented data models depend on the presence of a known and regular schema. Lore (Lightweight Object Repository) is a system developed to store and manage with the semi-structured data [10,11,13,14]. The data model used in Lore is called OEM (Object Exchange Model) [23,24].

XML, which is very similar to OEM, is a textual language designed for representing and exchanging data on the Web [12, 25-31]. It has its own query language XML-QL [32]. Nested, tagged elements are the building blocks of XML. Each tagged element has a sequence of zero or more attribute-value pairs and a sequence of zero or more subelements. These subelements may themselves be tagged elements, or they may be "tagless" segments of text data. Because XML was defined as a textual representation language, an XML document always has implicit order. The order may or may not be relevant but is nonetheless unavoidable in a textual representation. A well-formed XML document places no restrictions on tags, attribute names, or nesting patterns. An XML document can be optionally accompanied by a Document Type Definition (DTD), which is essentially a grammar for restricting the tags and structures of a document. An XML document satisfying a DTD grammar is considered valid.

In order to represent XML internally, the Document Object Model (DOM) has been defined [33] to enable XML to be manipulated by programs. DOM defines how to translate an XML document into data structures and can serve as candidate XML data model. While the DOM parser is tree based, another event-based parser, called SAX, was proposed by Megginson [34]. In our DTD Inference Engine, we use the SAX parser.

The differences between XML and OEM can be summarized as follows: unlike OEM, XML has attributes. Furthermore, the elements in XML are ordered while OEM has no internal order among its elements. Finally, OEM data is viewed as a graph while XML data forms a tree (if one does not count the ID references).

In order to facilitate the querying an OEM data in LORE, the Lore group has also developed a DataGuide [19,20]. The DataGuide is intended to be a concise, accurate, and

convenient summary of the structure of a Lore database. Lore formally defines a DataGuide for an OEM source S as an OEM object d such that every label path of S has exactly one data path instance in d, and every label path of d is a label path of S. The definition depends on the technical definitions of label path and data path instance. For additional details on Lore and DataGuide, please refer to Goldman and Widom [19].

The Lore DataGuide is similar to the DTD. They are both the summary of structural information of the source. The difference is that that DataGuide is a graph while DTD is a grammar for a context-free language. Because of its structure, it is awkward to express the DTD as a tree or graph. Instead we use a three-dimensional linked list data structure to represent the DTD in our DIE, which will be described in Chapter 6.

The Lore group has shown the existence of multiple DataGuides which correspond to a single source. We have similar results with DTD for XML documents. The same researchers also showed that a minimal DataGuide, which is unique, is not always desirable. Finally they have defined the notion of *Strong DataGuide* and have presented an algorithm to find a Strong DataGuide. This is similar to inferring a DTD for an XML document, as is the case in our research. Accordingly, we define notions of sound, tight and closure DTD and we will discuss the details in Chapter 5.

The Lore group also proposed the idea of incremental maintenance for their DataGuide. The idea is simple. The extraction of the DataGuide is time consuming. In those cases where the changes to the source are minor, it is more efficient to apply the changes directly to the DataGuide rather than extracting on from scratch. This depends on the change detection algorithms. This so-called incremental DataGuide maintenance

depends on the ability of the source to notify the DataGuide generation algorithm of changes. Hence, change detection is another active research area [35-42].

E. Myers gave the classic text change detecting and editing algorithms based on the Longest Common Subsequence (LCS) [35]. The GNU *diff* program uses this algorithm [36,37]. Change detecting for structured data other than text is more difficult. S. S. Chawathe et al. proposed an algorithm for change detection in hierarchically structured information [42]. We have proposed an algorithm for incrementally maintaining DTDs, which faces the similar issues to those, addressed by the DataGuide. Our maintenance approach is discussed in Chapter 7.

3.2 DTD Generators

The database group in the University of California at San Diego has conducted some studies on DTD inference [43]. The goal of their project is to develop a mediator architecture for XML data, which necessitates the need for being able to automatically generate DTDs for mediator views. Thus a central component of the mediator is a DTD inference module. They used the concept sound DTD and tight DTD, but in an intuitive, vague and undefined sense. We make some refinement on these concepts and give our own strict definition of sound and tight DTD and tighter DTD. Additionally we add the definition of closure DTD.

In 1995, the Online Computer Library Center (OCLC) launched the Fred project to provide some aid in authoring SGML documents. Fred is an extended Tcl/Tk interpreter and can generate SGML DTDs [41]. In the middle of December of 1999, Michael Kay published his SAXON DTD Generator. His motivation is similar with Fred,

to provide some authoring aid. Compared with Fred and SAXON DTD Generator, our DTD Inference Engine has three major advantages: first is the factorization reduction. For example, for an element declaration of the form

AX | AY | AZ | AU | BZ | BX| BU | BY | DY | DX | DU | DZ | CZ | CX | CU | CY,

Fred does no reduction. It simply returns a juxtaposition of all sections. We call this lazy juxtaposition. When there are many sections, like in the periodic table, you may have more than one hundred sections. Without any simplification, it is hard to grasp the internal structure of the XML data. At the other extreme, Michael Kay's DTD Generator does a lazy collapsing, and the result will become

(A | B | C | D | X | Y | Z | U)*.

Although this is a sound DTD, a large amount of original information is lost. The order is totally wiped out. Recently IBM delivered DDbE in its Alphaworks. DDbE performs a one step factorization. For the above example, DDbE gives a result as

A, (X|Y|Z|U) | B, (Z|X|U|Y) | D, (Y|X|U|Z) | C, (Z|X|U|Y)

Our DIE will output the DTD as

(A | B | C | D), (X | Y | Z | U),

which is closest to the original structure.

Second, it handles multiple documents. Both the Fred and SAXON DTD Generator take one single XML document as input. We consider the situation where there are multiple documents conforming to a single DTD.

Finally, it applies incremental maintenance of the DTD. There are situations that the XML source documents changes dynamically and in some application domains, the change may be fast. The DTD has to keep pace with the source documents. When the

change is small, it is more efficient to maintain the DTD by the source change, rather than extracting the entire DTD from scratch. Our approach is based on a log of changes.

3.3 Theoretical Studies on DTD Inference

Theoretical study on DTD inference is also active. S. Nestorov et al. studied the aspect of extracting schema from semi-structured data [44]. D. Angluin conducted extensive research on the inference of deterministic finite automata [45].

DTD in essence is the grammar of XML language. The study of DTD inference falls in the category of formal languages. While the authors claim in their paper [43] that the languages specified by the DTDs are restricted to regular languages, we believe a DTD grammar specifies a broader language, namely the context-free language [22,46,47]. According to this analysis, we designed our tree-dimensional linked list data structure to represent DTDs.

CHAPTER 4
XML SPECIFICATION PERTAINING TO DTD

The official specification of the XML language proposed by W3C forms the basis for the DTD inference algorithm described in this thesis. Our DTD inference engine is based on the Extensible Markup Language (XML) Version 1.0 from 1998 proposed by W3C [12]. A thorough understanding of the specification of the syntax and semantics pertaining to the DTD is essential to understanding the implementation of the DTD Inference Engine. This chapter gives a synopsis of annotations of the W3C XML specification pertaining to the DTD. We provide the Extended Backus-Naur Form (EBNF) specification as it pertains to the DTDs in the appendix of this thesis. Some syntactical categories, which have to do with the general XML language, such as Name, Nmtoken, space, Processing Instructions, comments, etc. have been omitted. For those, the reader is invited to consult the W3C specification [12].

A DTD consists of a document type declaration section followed by markup declarations. The DTD begins with "<DOCTYPE" followed by a Name and a "[", and ends with "]>." The beginning and end are usually placed on separate lines. This is known as a document type declaration. It is important to note that the name in the document type declaration must match the element name of the root element.

The markup declarations go between the square brackets ("[]"). There are four types of markup declarations in XML: element type declarations, attribute list declarations, entity declarations and notation declarations.

4.1 Element Type Declarations

Element type declarations identify the names of elements and the nature of their content. Element declarations describe the logical structure of the document. If the element is empty, we use the keyword EMPTY. If we use the keyword ANY, then we impose no restriction on the structure of this element. Generally we list the children of the element in the element declarations. Note that the logical structure of an XML document represents an ordered tree. This means that the order among the children matters. An element "<!ELEMENT name (FirstName, LastName)>" is considered different from an element "<!ELEMENT name (LastName, FirstName)>", although in many applications, order is not important. In those cases, XML provides additional structural information and is more restrictive.

We can also have the #PCDATA type and children as a mixed content case in the element declarations. The children list in the element declaration may be specified as a disjunctive list in which children are separated by vertical bars "|". This is the notation commonly used in the EBNF grammars [22].

Wild cards common to regular expressions are also used. The sequence "A?" matches zero or one occurrence of "A." "A+" matches one or more occurrences of "A" and "A*" matches zero or more occurrences of "A."

4.2 Attribute List Declarations

An attribute is a name-value pair that is used within the start tag of the element to describe the nature of the element. Attribute list declarations identify which elements may have attributes, what those attributes are and what values the attributes may hold, including possible default values.

The attribute list declaration consists of the element name, followed by a list of attribute name, type and possible default values. There are ten possible attribute types: CDATA, Enumerated, NMTOKEN, NMTOKENS, ID, IDREF and IDREFS, ENTITY, ENTITIES, NOTATION and an Enumerated NOTATION. Below we provide a brief explanation of these types.

CDATA: CDATA is the most general attribute type. It means the value may be any string of text that does not contain a less than sign (<), ampersand (&), or quotation marks (“).

Enumerated: The enumerated type is not an XML keyword. Instead it is a list of possible values for the attribute, separated by vertical bars, as in <!ATTLIST p visible (true | false) “true”>.

NMTOKEN: The NMTOKEN attribute type is a restricted form of a string attribute. It restricts the value of the attribute to a valid XML name, which must begin with a letter or an underscore (_). Subsequent letters in the name may include letters, digits, underscores, hyphens and periods.

NMTOKENS: The NMTOKENS attribute type is the plural form of NMTOKEN. It allows the value of the attribute to be composed of multiple XML names, separated from each other by white space.

ID: The ID type uniquely identifies the element in the document. An attribute value of type ID must be a valid XML name. All of the ID values used in a document must be different. A particular name may not be used as an ID attribute of more than one tag. Furthermore, each element may not have more than one attribute of type ID. Typically, IDs exist solely for the convenience of programs that manipulate the data.

IDREF and IDREFS: The IDREF type allows the value of one attribute to be an element found elsewhere in the document. The value of the IDREF attribute must be the ID of an element elsewhere in the document. Specially, the IDREF attribute value must be identical to the value of an ID attribute in another element. The value of IDREFS attribute may contain multiple IDREF values separated by white space.

ENTITY: An ENTITY type attribute enables one to link external binary data--an unparsed entity--into the document. The value of the entity attribute is the name of an external parameter entity declared in the DTD that links to the external binary data.

ENTITIES: ENTITIES is a plural form of ENTITY. An attribute of type ENTITIES has a value part that consists of multiple entity names separated by white space. Each entity name refers to an external binary data source.

NOTATION: The NOTATION attribute type allows an attribute to have a value specified by a notation declared in the DTD. One can use this type to specify the preferred helper application for an unparsed entity.

We can also specify default values using one of the four different ways to make restrictions on the default values.

Specify the default value: An attribute can be given any legal value as a default value. The attribute value is not required on each element in the document. If it is not present, it will appear to be the specified default value.

#REQUIRED: Instead of specifying a default value, you may use the #REQUIRED keyword to enforce that the value of this attribute must be provided for this element in the document, although the attribute of the same element may take different values on different occurrence of the element with the same name.

#IMPLIED: In this case, the attribute value is not required, and no default value is provided. It says that providing the value to this attribute is optional. If a value is not specified for this attribute, the XML processor must proceed without one.

#FIXED: The attribute declaration specifies that an attribute has a fixed value. In this case, the attribute is not required, but if it occurs, it must have the specified value. If it is not present, it will appear to be the specified default.

4.3 Entity Declarations

Entity declarations allow one to associate a name with some other content fragment. That content fragment can be a block of regular text, a document type definition or a reference to an external file containing either text or binary data.

Entity references are classified as either general entity/parameter entity references, or internal/external entity references. For the internal entity references, the entity is defined and used in the same file, while for the external entity reference, we use the name, which is the abbreviation of the entity that is physically stored in another file.

Internal General Entity Reference: A general entity reference is an abbreviation for commonly used text. The general entity reference is declared as follows: <!ENTITY name “replacement text”>. The name is the abbreviation for the entity. Whenever the abbreviated name appears in the document, it is replaced by the text declared in the entity declaration. However, to use general entity references in the DTD, several restrictions apply. First, the statement cannot use a circular reference. Second, the declaration of the reference must come before any use of the reference. And the third, general entity references may not insert text that is only part of the DTD and will not be used as part of the document content.

Internal Parameter Entity Reference: A parameter entity references are very similar to general entity references, except that parameter entity references begin with a percent sign (%) rather than an ampersand, and parameter entities can only appear in the DTD, not the document content. Parameter entities are declared in the DTD similar to general entities, but with the addition of a percent sign before the name: <!ENTITY % name “replacement text”>.

External General Entity Reference: An external entity associates a name with the content of another file. External entities allow an XML document to refer to the contents of another file. External entities contain either text or binary data. If they contain text, the content of the external file is inserted at the point of reference and is parsed as part of the referring document. Binary data is not parsed and may only be referenced in an attribute. Binary data is used to reference figures and other non-XML content in the document. <!ENTITY name “URL”>.

External Parameter Entity Reference: An external parameter entity is similar to an internal parameter entity in that the external parameter entity reference also begins with a percent sign (%) and can only appear in the DTD, not the document content. The difference is that the external parameter entity is in another file and in the declaration, the key word SYSTEM is used and the URL of the external parameter entity is specified, using the syntax, <!ENTITY % name SYSTEM “externalID”>.

4.4 Notation Declarations

In practice, the notation declaration allows us to specify some helper applications to process the unparsed entity. We can link to an unparsed entity through an external general entity reference. In addition, we include the NDATA keyword and the type of the data in the entity declaration. For example, to associate the entity reference &logo; with the GIF image http://sunsite.unc.edu/javafaq/logo.gif, one places the following declaration in the DTD:

<!ENTITY logo SYSTEM http://www.w3.org/xml/logo.gif NDATA gif>

Each unparsed entity is associated with a notation. In theory, the notation is the format of the non-XML data. A notation is a set of rules that the data follows, which are generally quite different from the rules that XML data follows. In practice, these rules are merely the name of a program that understands the data format involved. For example, the declaration <!NOTATION gif SYSTEM “Image Viewer”> says that data notated with the notation “gif” may be passed to the “Image Viewer” application for processing. The parser simply passes that data along to the application (which is free to ignore it).

The rule for entity references is that they have to be declared in the DTD before they are used in the document. This means, that if a document does not have a DTD, then it cannot use any entity references and it must be a stand-alone document. Hence in the implementation of our DTD Inference Engine, we do not have to consider the inference of entity declarations and notation declarations. The two major parts that are relevant to the engine are element declaration and attribute list declaration.

CHAPTER 5
DTD INFERENCE AND CONTEXT-FREE LANGUAGES

5.1 What Kind of DTD Is Desirable?

So far, we have motivated the need for DTD inference to describe the source XML document, and we have introduced the reader to the XML language specification, which pertains to DTDs. However, before we can start implementing our DTD Inference Engine, we have to define precisely what our objective is. For example, we cannot neglect the fact that a DTD describes a class of many (possibly infinitely many) XML documents and that one particular XML document is just one instance, or one snapshot, of the infinitely many documents that conform to a particular DTD. What kind of DTD do we want to infer for a particular XML document? What is considered "a correct" DTD? During this and the next sections we point out that correctness does not make much sense. Instead, we adopt the term "sound DTD." In such a situation we need to handle the opposing desires of "accuracy" versus "conciseness." In one extreme, we could use the keyword "ANY" in the element declaration. That is an extremely concise description but one that does not convey any useful information about the document structure. In the other extreme, we can include every detailed piece of information that is contained in the document in the DTD such that the source document is the only document that conforms to that DTD. In this case, the DTD is accurate but fails to be

concise enough to serve as a summary. As part of our work we have to find a reasonable compromise between the two extremes.

A first observation is that the DTD can be considered a context-free grammar describing the correct schema of the corresponding XML documents. Hence the XML document is a derivation of this grammar: the tags correspond to the non-terminal symbols (or syntactic categories) of the language, while PCDATA corresponds to the terminal symbols. It is worth noting that some papers mistakenly refer to the tags as the alphabet [43]. In the context of DTD inference, the PCDATA or the terminal symbols are not important.

To carry this analogy further, one can say that the XML document forms a derivation tree, or parse tree of a context-free language described by the context-free grammar represented by the DTD. When stripped of its markup tags, the remaining XML document forms one word (or string) in the context-free language.

A parse tree is a concept used in language theory but may cause some confusion here because of the use of the term “XML parsers.” There are many XML parsers available but XML parsers are different from traditional parsers used in programming languages. The parser for a programming language takes a word (program) as input and builds a parse tree (or derivation tree) using the known grammar. If the grammar is ambiguous, the parse tree may not be unique for a particular word (program). That is why programming languages require unambiguous grammars. However, in the case of an XML document, the XML document itself is the parse tree (derivation tree). The parse tree is already there, only in the text format. The grammar is unknown if the DTD is not given. The XML parser takes the parse tree in text format and converts it to an equivalent

parse tree in a format that is usable for computer application programs. To do this, the XML parser does not need the grammar. In order to avoid this confusion, henceforth, we shall use the term derivation tree instead of parse tree.

Since a DTD is a grammar, we do not represent it as a tree. There are some papers [17,43], which represent DTDs as a tree or a graph. We believe this is awkward and inefficient. In fact, it is not a tree because a DTD contains regular expression wild cards "*", "+" and "?", and also because each non-terminal symbol can have multiple productions like A | B | C | D. Instead we use a multi-dimensional linked list structure to represent the DTD in our implementation, which is discussed in Chapter 6.

To summarize, a DTD is a context-free grammar using a particular syntax specified by the W3C, which is different from the syntax of a well-formed XML document bodies. Contrary to the DataGuide, which is a graph and can be considered a tree in special cases, a DTD is not a tree.

With this in mind, we can now define our task precisely: given a derivation tree T for a certain unknown context-free grammar, we are to find a context-free grammar G such that T is one of the derivation trees of grammar G. Obviously such a grammar is not unique and we need to overcome a certain amount of arbitrariness. In the next chapter, we give some rules for the DTD inference. However, before we can define a reasonable set of rules, we need to investigate the problem in the context of language theory first.

5.2 Kernel Derivation Tree

Let us consider the derivation tree DT for a context-free grammar G. Terminal symbols are represented by squares while non-terminal symbols are represented by circles. We view such a tree as an extended tree. That is to say, we view the leaves of the derivation tree, which are the terminal symbols, as external nodes while the non-terminal symbols as internal nodes. We call the tree containing only non-terminal symbols a Kernel Derivation Tree or KDT for the source XML document. It is obvious that this KDT is our only concern in the DTD inference process. From now on, we will concern ourselves with a KDT instead of the full derivation tree.

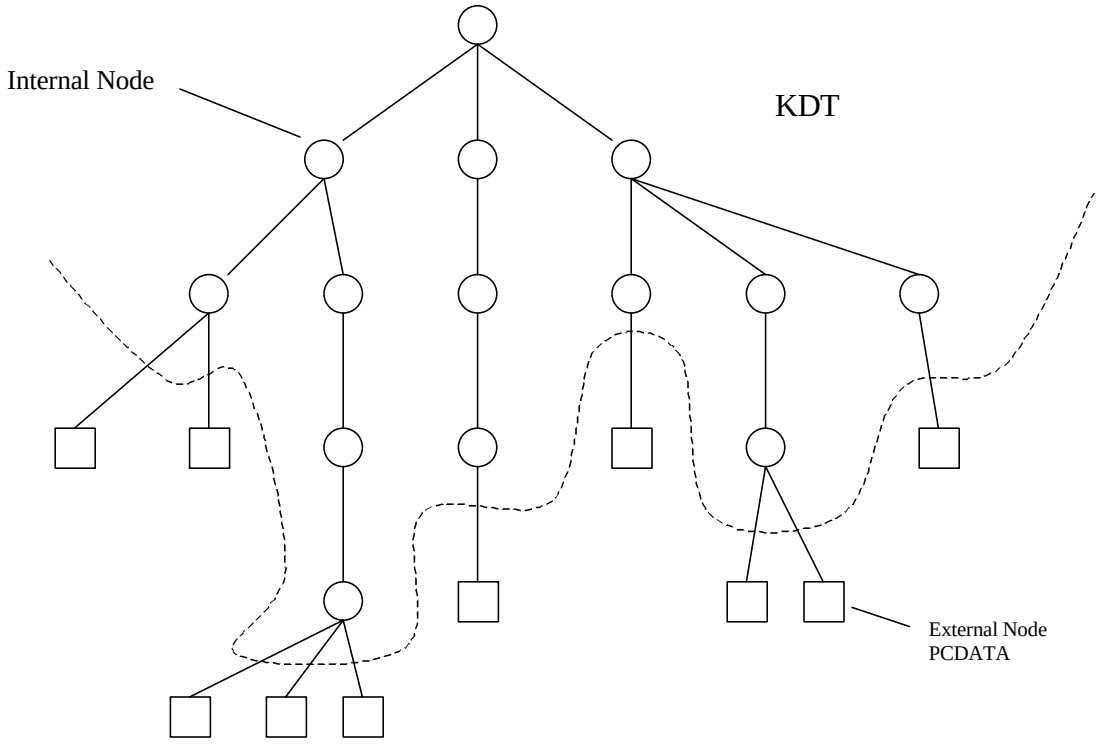

Figure 2. Kernel Derivation Tree

5.3 Multiple Derivation Trees for a Given Grammar and Multiple Grammars for a Given Derivation Tree

The term "word" is used in language theory to denote a finite string of terminal symbols from a certain alphabet. We know each word has a KDT but different words may have the same KDT. We now investigate how many derivation trees can be derived from a given grammar and how many grammars can generate the same given derivation tree. Furthermore, we will examine the relationships between grammars that can generate the same given derivation tree. To answer these questions, we provide the following two theorems.

THEOREM 1: A finite language has a finite number of KDTs.

The proof is straightforward. A finite language has a finite number of words. Each word has a finite number (in an unambiguous language, just one) of KDTs. So a finite language has a finite number of KDTs.

From this we can deduce that if we generate a DTD from a source XML document and the DTD describes a finite language, there must be at most a finite number of XML documents besides this source XML document that conforms to the same generated DTD. Thus we have bound for the number of candidate XML documents.

THEOREM 2: An infinite language has an infinite number of KDTs.

Proof: We use a proof by contradiction. Suppose the opposite of the proposition is true. That is, the language is infinite but has a finite number of KDTs. Then, grammar is always a finite description of a language. No matter whether the language is finite or infinite, the grammar has a finite set of terminal symbols, a finite set of non-terminal

symbols, and a finite number of productions. When parsing a word, the substitution of the leaf nodes of the KDT only involve productions of the form: <non-terminal> ::= *abc*.

That is, the right hand side only contains terminal symbols. Let *R* be the maximum length of the right hand side string of terminal symbols in all of this type of productions. If there are a finite number of KDTs, there is a KDT with the maximum number of leaves. Let this number be $l_{\max}$. Therefore this language can generate a string no longer than $l_{\max} \bullet R$. Hence this language is finite, which contradicts the assumption that the language is infinite.

From this we can deduce, that if the XML document requires a DTD that describes an infinite language, then there are an infinite number of XML documents besides this one that also conform to this DTD. This XML document belongs to an infinite family. Even if we restrict ourselves from using the Kleene star *, a simple XML document may still require to be fitted into an infinite language. Here is a simple example of such an XML document that is a grammatical markup of the English phrase “Lady with Flower with Ladybug”, as shown in Figure 3. This can be described by the following minimal DTD:

<!ELEMENT NOUN_PRASE (NOUN | NOUN, PREP_PHRASE) >

<!ELEMENT PREP (#PCDATA) >

<!ELEMENT PREP_PHRASE (PREP, NOUN_PHRASE) >,

which describes an infinite language.

```
<?xml version="1.0"?>
<NOUN_PHRASE>
  <NOUN>Lady</NOUN>
  <PREP_PHRASE>
    <PREP>With</PREP>
    <NOUN_PHRASE>
      <NOUN>Flower</NOUN>
      <PREP_PHRASE>
        <PREP>With</PREP>
        <NOUN_PHRASE>
          <NOUN>Ladybug</NOUN>
        </NOUN_PHRASE>
      </PREP_PHRASE>
    </NOUN_PHRASE>
  </PREP_PHRASE>
</NOUN_PHRASE>
```

Figure 3. An XML Example

5.4 Sound, Tight and Closure DTDs

We now give definitions of additional terminology, which is needed to analyze the rules of DTD inference.

Definition 1. **Sound DTD.** Given *S*, a set of XML documents with the same root name, and a DTD *D*, which is equivalent to a context-free grammar *G*, if all the document trees in *S* are the KDTs of the grammar *G*, then *D* is called a sound DTD of *S*, or *D* is sound with respect to *S*.

Note that set *S* can be a singleton set. That is *S*, can be a single XML document.

Definition 2. Given *S*, a set of XML documents with the same root name and a sound DTD *D* of *S*, if all the KDTs of *D* are in the set of the document trees of *S*, then *D* is called a tight DTD of *S*, or D is tight with respect to *S*.

Definition 3. If D_1 and D_2 are sound DTDs of *S*, a set of XML documents, and G_1 and G_2 are the corresponding Grammars of D_1 and D_2 respectively, G_1 and G_2 have the same set of non-terminal symbols, and $L(G_1) \subset L(G_2)$, where L(G1) and L(G2) denote the languages generated by G_1 and G_2 respectively, then D_1 is called tighter than D_2 with respect to *S*.

This relationship is illustrated in the diagram depicted in Figure 4.

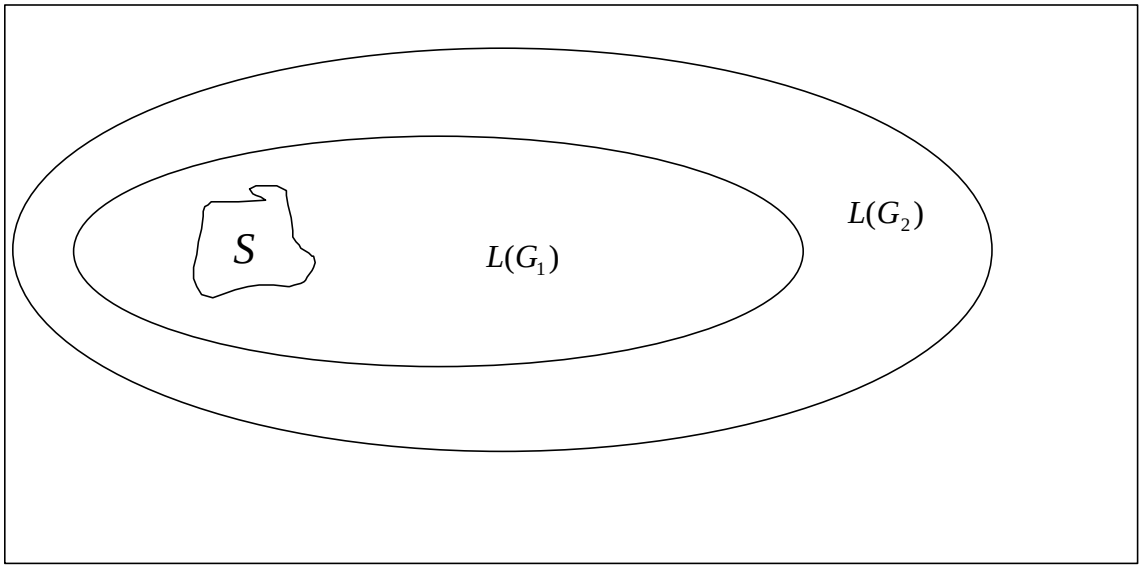

Figure 4. $L(G_1)$ Is a Tighter DTD for S than $L(G_2)$.

Definition 4. **Closure DTD.** If *Cl* is a sound DTD of XML document *S*, and if *D* is any different sound DTD of *S*, then CL is tighter than *D*, then *CL* is called the Closure DTD of *S*.

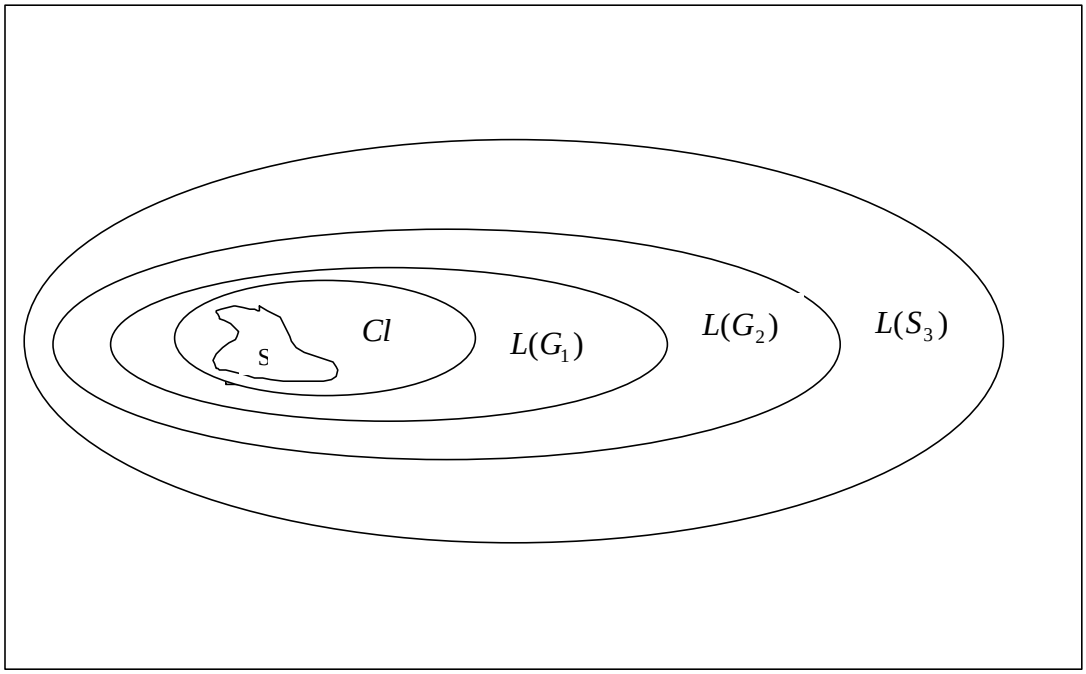

Figure 5. Closure DTD

With these concepts defined, it is easy to draw the following conclusions: the tight DTD is also the closure DTD but the closure DTD of an XML document may not be tight. However, it is tighter than any other sound DTD. An XML document may neither have a tight DTD nor a closure DTD as illustrated in the diagram shown in Figure 6. If an XML document S has a sound DTD that corresponds to a regular grammar, then S has a Closure DTD because the intersection of two regular languages is still a regular language. If the KDT of an XML document S is recursive (either directly or indirectly), then the language is infinite and therefore S has no tight DTD. An XML document without recursion can always be described by a finite language, but this may not always be desirable.

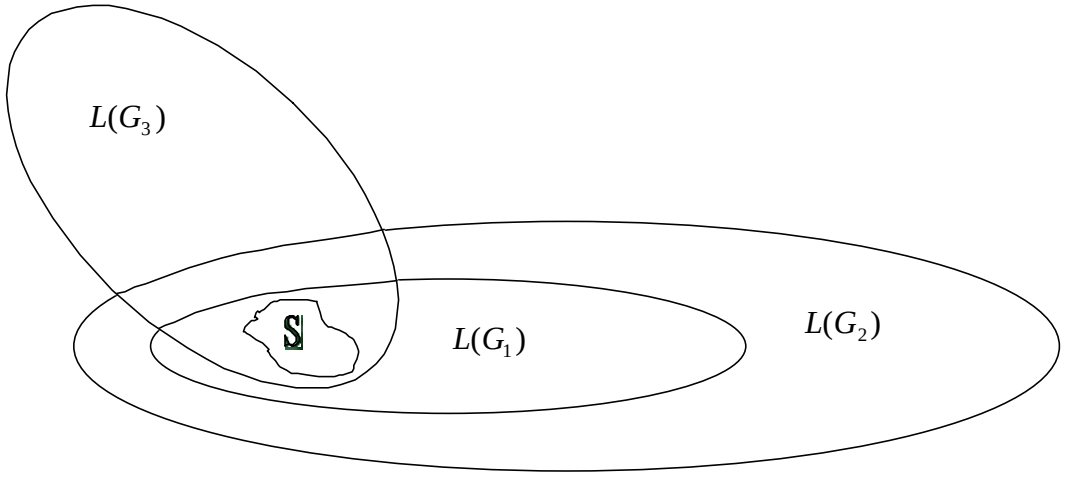

Figure 6. XML Document without Closure DTD

Tightness is not important in the DTD inference problem and in many cases there does not exist a tight DTD. Closure DTD is more desirable but is still not absolutely necessary. Sometimes we can relax this restriction a little bit.

5.5 DTD Reduction

Give a sound DTD, a sequence of reductions is still possible in order to simplify the expressions. There are two kinds of reductions:

1. Equivalence reduction: The DTD D_1 is changed to a different form D_2 but still describes exactly the same language.

$L(D_1) = L(D_2)$.

e.g. D_1: <!ELEMENT TERM (AB| AC)>

D_2: <!ELEMENT TERM (A (B | C))>

2. Relaxing reduction: The DTD D_1 is changed to a different form D_2 but now

$L(D_1) \subset L(D_2)$.

e.g. D_1: <!ELEMENT TERM (AA | AAA)>

D_2: <!ELEMENT TERM (A*)>

In our design of the DTD Inference Engine, we use both the equivalent reduction and relaxing reduction. Our so-called factorization reduction is analogous to an equivalence reduction and our degeneration reduction is analogous to a relaxing reduction, as is shown in the next chapter.

CHAPTER 6
THE DTD INFERENCE ENGINE

So far, we have discussed the properties of DTDs and what kinds of DTDs are desirable. We have seen that tightness is not absolutely necessary but we may lose information if the DTD is loose. However, we still need to discuss several underlying assumptions for implementing the DTD Inference Engine. These assumptions are formulated as the rules. In the following sections, we first decide on the rules we are going to use and then discuss their implementation of the inference engine.

6.1 Rules of DTD Generation and Reduction

The role of the DTD is twofold. One is to restrict the allowable structure of the document. The second is to provide a summary of the document structure information. In the case when the original document has no DTD, the inferred DTD has no real restricting power over the original document. However, it still provides structural summary information. We have seen from the discussion and analysis in the last chapter that a tight, or even a closure DTD is not always desirable or possible. However, as a minimum requirement, it has to be a sound DTD but there may be many sound DTDs for a single document. Our goal is to obtain an intuitive DTD. As a result, there is still a lot of room in determining what DTD to generate. What we hope to achieve is to make our inferred DTD resemble the missing (hypothetical) DTD written by the creator of the

document as closely as possible. Furthermore, it should capture the basic structural information of the document. We have seen that tightness is not absolutely necessary. However, if the DTD is loose, it loses information in the document structure. Of course, there are some factors in the DTD, especially in the attribute list declarations, which are purely based on the author's intentions. They are semantic, rather than syntactic and impossible for the DTD inference engine to infer.

We now list the rules we have adopted for guiding our DTD Inference Engine to generate DTDs in the spirit of the above discussion.

6.1.1 Rules for Element Declarations

The first five rules follow from the XML specification in a straightforward manner. Rule 6 through Rule 9 reflect our policy on Kleene stars and reductions, which may vary among different implementations of the DTD inference engine.

RULE 1. ANY Rule: Do not use ANY under any circumstances.

This rule does not need an explanation because although ANY is a legal DTD syntax construct, it hides the information provided by the XML document and leads to ambiguous DTDs.

RULE 2. EMPTY Rule: If the element Z has no children, use <!ELEMENT Z EMPTY>.

RULE 3. PCDATA Rule: If the element Z contains only parsed character data, use <!ELEMENT Z (#PCDATA)>.

RULE 4. Simple Sequence Rule: If the element Z only has one occurrence in the document, and has the child sequence A, B, C, D, E, use <!ELEMENT Z (A, B, C, D, E)>.

RULE 5. Section Rule: If the element Z occurs twice (or more times), and if the sequence of children in the first occurrence is A, B, C, D and the sequence of children in the second occurrence is P, Q, R, make two sections separated by the vertical bar (OR). <!ELEMENT Z (A, B, C, D | P, Q, R)>. This will be reduced further using the rules introduced below.

RULE 6. Kleene Star Rule: if two or more children with the same name are next to each other, use Kleene star.

For example, if we have A, A or A, A, A, use A*.

The rational behind using * instead of + is that, by doing this we get a more general, looser DTD but we know the inferred DTD has no restrictive power. The same element might appear in another instance of document with the same element without child A. Or, if we suppose the source dynamically changes, "A" might be deleted in the future. In that case, we do not have to update the DTD to keep it in accordance with the source. We discuss DTD maintenance in next chapter.

RULE 7. Reduction Rule--Subsequence Rule: suppose we encounter two occurrences of the same element and we have a sequence of children for each occurrence. If one child sequence is the subsequence of the other, we merge the two sections into one and use the Kleene star where one child does not appear in the subsequence.

For example, if on one occurrence of element X, we see child sequence A, M, P, T, K, Q; on another occurrence of X, we see child sequence M, T, K, where M, T, K is a subsequence of A, M, P, T, K, Q. Using Rule 7, we merge the two sections into one as A*, M, P*, T, K, Q*.

RULE 8. Reduction Rule--Factorization: Take out the common factors among the sections separated by vertical bars.

For example, AX | AY| BX | BY will be reduced to (A|B), (X|Y). This gives us more concise and intuitive DTD. As discussed before, this is an equivalent reduction.

RULE 9. Reduction Rule--Degeneration: After all other reductions have been completed, if there are still too many sections left, cull all the children names in all the sections (the union of all the sections as sets, without considering the order) and collapse them into the unordered form using Kleene star. For example, if we have A, B, D, C | B, A, C | D, B, C, we can degenerate them into the form (A | B | C | D)*. We need to set a threshold of the number of sections over which we will apply the degeneration rule. In the implementation of this thesis, we set the threshold to be 10. This is to avoid too long a child list. This number 10 is subjective and arbitrary. Different people may want choose a different threshold, like 15 or 20.

6.1.2 Rules for Attribute List Declarations

For the attribute declaration, we need to find the attribute name, type and default value or default type like #REQUIRED, #IMPLIED or #FIXED. In the #FIXED case, we also need to supply the fixed value. There are ten attribute types. We rely on the XML parser to report attribute types. Unfortunately, the XML parser relies on the DTD to report attribute types. Without a DTD, the XML parser will just report CDATA as the type. One solution is to guess a type. For example, if we see a space in the attribute value, we would report CDATA. If there is no space, then report NTOKEN. If the value is unique for each occurrence, then report ID as the type. However, we strongly believe that the type is the author's semantic intention rather than the syntactic structure. An attribute

value without any intervening space in between could well be intended by the author to be CDATA, instead of NTOKEN. All the IDs have unique values. However, the attribute with all unique values may not be intended to be IDs. We do not believe that a guess of semantic intentions using syntactic structures as clues is wise or useful. As a result, we decided instead to treat them as CDATA.

As for the default value type, we adopted the following rule: if the attribute appears in all the occurrences of an element, we mark it as #REQUIRED. If it is missing in some of the occurrences, we mark it as #IMPLIED. Among the #REQUIRED attributes, we further check its values in all the occurrences. If the values in all the occurrences are the same, and the total number of occurrences exceeds a given threshold, we mark it as #FIXED followed by the value. The threshold we used in this implementation is 5. Again this is arbitrary. Different people may want to choose a different threshold but it does not matter as long as it is in a reasonable range. The rational for this is as follows. If we see a different value in all the occurrences, it certainly does not qualify as #FIXED. However, even if the values are the same in all the occurrences, but if the number of occurrence of this element is small, say it only appears twice, we are not sure this attribute will always take the #FIXED value on all instances, because the current document is only a snapshot of a more general structure. Although the attribute list declaration appears to be simple to implement, it still requires an important breakthrough as described in the algorithms discusses in later sections.

6.2 Data Structures Representing the DTD

The data structures used to represent the DTD are essential to an efficient DTD inference engine. As we have argued before, we do not represent the DTD as a tree structure. Instead the data structure we use is a three dimensional linked list, shown in Figure 7. The top-level list is the list of elements represented by nodes we call

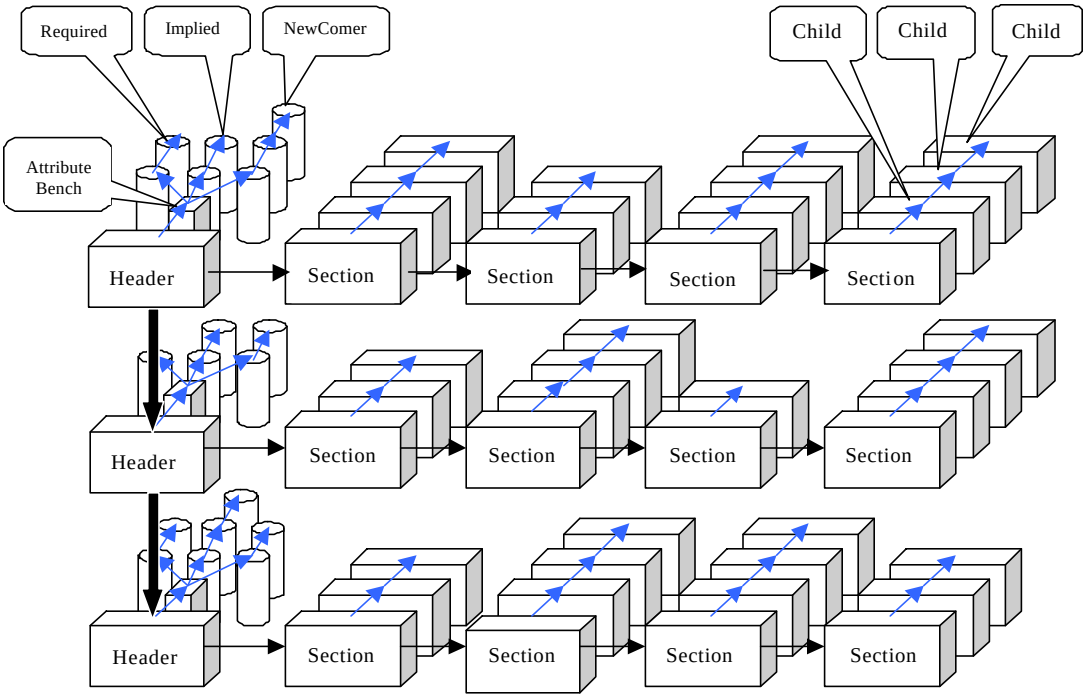

Figure 7. Three-Dimensional Linked List

elementHeaders. Each element contains a list of Sections, which in turn contains a list of children, which we call elementNodes. All the children of one element are placed in

different sections, just like the children are separated by vertical bars in the textual representation. The need to separate children in different sections to make reduction easier. Each elementHeader, Section or elementNode is a node just a regular node in a linked list, except it may have more private fields. Those are boolean flags to record status, as well as static integer values for those threshold values, e.g., boolean degenerated, int degenerateThreshold (in elementHeader), boolean leftFactorized, boolean rightFactorized (in Section), and boolean potentialStar(in elementNode).

We choose a linked list rather than a hash table although we have to perform a lot of lookups. The reason is that although we can have better efficiency for lookups if using hash table, we also have frequent traversals, which is not convenient using hash table. More over, the DTD usually is small, even if the document is big. This favors a three-dimensional linked list structure.

We also link the attribute part to each elementHeader. We call this AttributeBench because this is actually the workbench, or work place to manipulate the attributes. AttributeBench is divided into three parts: a Required section, an Implied section and a NewComer section. The Required section is intended to hold attributes with the #REQUIRED default type and Implied section is intended to hold attributes with #IMPLIED default type. When a new attribute is added to the AttributeBench of this element, it is placed into the NewComer section. Then it will need many complicated juggles among the attributes in the three sections, just to partition all of the attributes into the Required section and Implied section. And finally we will split a part from Required as the #FIXED, with the aid of some private flag fields in the data structure. We do not make an explicit section for #FIXED attributes.

6.3 Overview of the Architecture of the DTD Inference Engine

The DTD inference engine has three major components: the Element Engine, the Attribute Engine, and Reduction Engine, shown in Figure 8. We also have a File Handler sitting in the front. We briefly describe the functionalities and interactions among the component.

The engine uses one or multiple XML documents as input. The File Handler handles the multi-document case. It checks if the root names of all input documents are the same, strips off the XML declaration header and generate a single super XML document. In the case of a single XML input document, the document bypasses the File Handler.

The Element Engine builds the element declaration part of the DTD. It receives an event report from the SAX parser and gathers element structural information while the traversing of the document.

The Attribute Engine builds the attribute list declaration part of the DTD. It manipulates the attributes for the default type information. The manipulation process is similar to the juggling; hence we call it the juggler.

When the engine has finished the traversal of the document, the DTD is built. At this point we may still want further reduction and simplification. As pointed out before, reductions can be both equivalence reduction, like sort and factorization, or relaxing reductions, like degeneration. As a result, the Reduction Engine starts when the end of

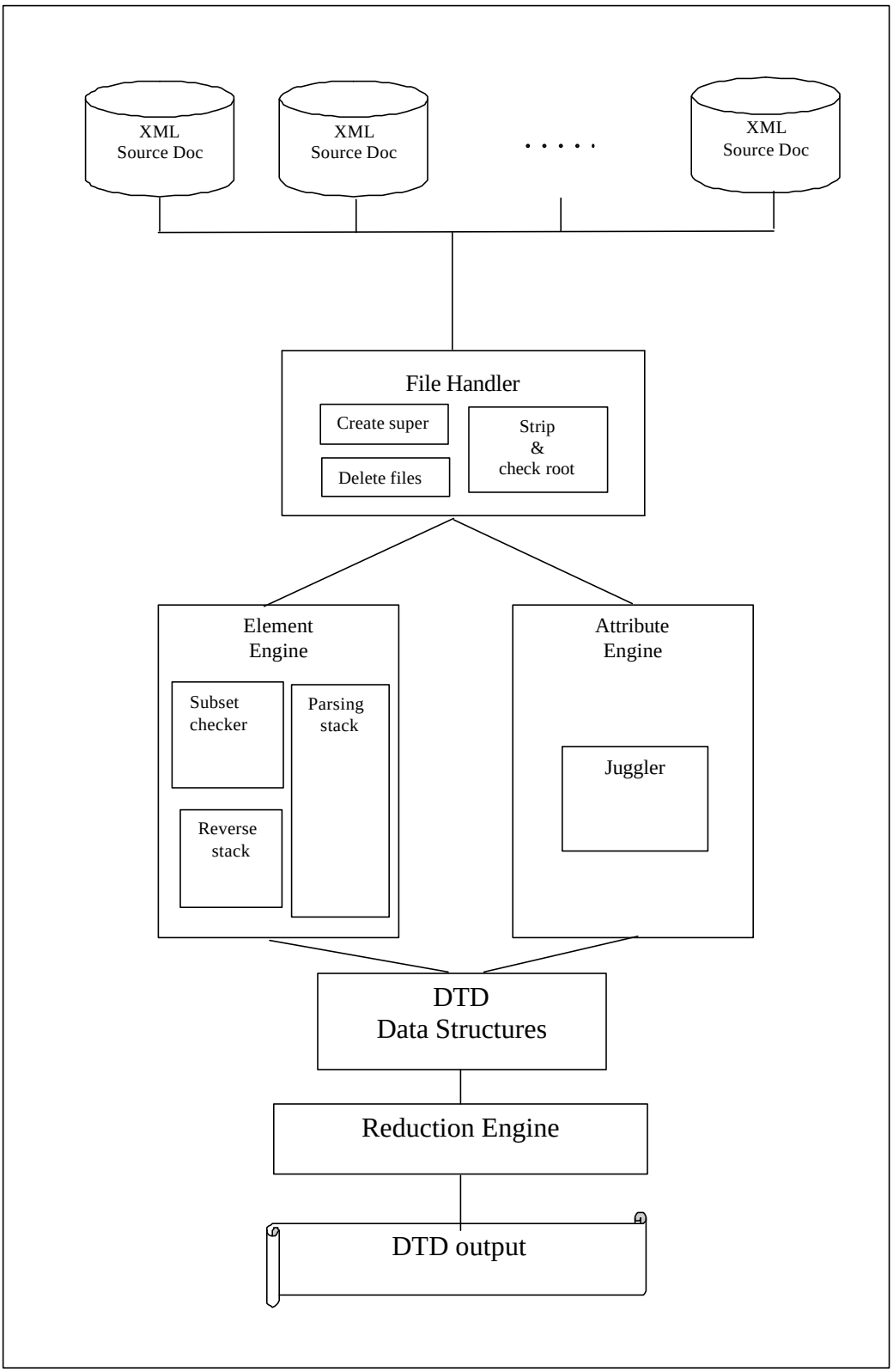

Figure 8. Architecture of the DTD Inference Engine

the document has been reached. After the reduction, we output the DTD in text format, which needs a traversal of the DTD data structure because all the subelements and attributes are in the linked lists.

6.4 Algorithms and Implementation

We have seen the overall architecture of the DTD inference engine. In this section we provide a detailed discussion of the algorithms we have used in implementing the various parts of the DTD inference engine. Our implementation is based Oracle's version of the SAX parser. We start with the single document input case. After this we discuss the multiple document input case and the use of the File Handler as shown in Figure 8.

6.4.1 Element Engine

The element engine infers the element declarations in the DTD. It takes the XML document as input. The SAX parser parses it and reports events to the DTD Inference Engine as follows: In the event of start-element, the Element Engine pushes the name of the element on to the parsing stack, and then pushes a string "Start" on the parsing stack, which will be used as a signal when popping from the stack later. Then it checks if the name is #PCDATA. If it is not #PCDATA but a name of a child element, it searches the ElementHeader list to see if the name is already in the list. If the name is not in the list, the Element Engine will append a new ElementHeader for this element name. We do not make a header for #PCDATA.

In the event of end-element, the Element Engine starts popping the stack until it sees the signal "Start". It pushes every element onto the reverse stack immediately after it is popped out of the parsing stack. We do this because when pushed onto the parsing

stack and popped out, the order of the elements is reversed. However, since the order is important in XML, we use the reverse stack to rearrange the elements into the original order. When the parsing stack stops popping, all the section of elements is now in the reverse stack. Now we pop the reverse stack and append the elements to the last section in that ElementHeader.

When the SAX parses the document, it proceeds in a depth first order through the document tree. We use push and pop operations on the parsing stack in this depth first traversal so that we can find the children of each element.

After appending the new section of children as the last section to the ElementHeader, we initiate immediate reduction, i.e., subsequence checking (or subset checking) to see if either the last section is a subsequence of a previous section or a previous section is the subsequence of the last section. In either case, Kleene star may be added in the whole sequence and the subsequence is deleted. This is either the last section or the precious section. For example, if one section is A, B, C, D, E and another B, C, E, then the latter is a subsequence of the former. We then change the section representing the full sequence A, B, C, D, E to A*, B, C, D*, E and delete the section representing the subsequence. We also have checking mechanisms to make sure that the Kleene star is not added more than once to an element. A special case of subsequence is an identical redundant section. In case the last section is identical to a previous section, the last section is deleted. Next we discuss the Attribute Engine.

6.4.2 Attribute Engine

The attribute list is divided into three sections: Required, Implied, and NewComer. In the three dimensional linked list data structure for the DTD, each ElementHeader has a field AttributeListBench. The AttributeListBench is intended as the

workbench or work-place to manipulate the attribute list. The AttributeListBench has three fields, Required, Implied, and NewComer. These are three linked lists of the same type, MyAttributeList. The node in each list is of type MyAttributeNode. We have developed our own MyAttributeList class instead of using or extending the AttributeList interface in SAX or the AttributeListImpl in the Oracle implementation. The reason is that, in the Oracle implementation, we have access to the implemented public methods but we do not have access to the individual nodes. We need other methods other than the provided. It would not be efficient to implement our methods only using their public methods without accessing the individual nodes and the pointers. In addition, we also wanted to add more fields to the node to which we do not have access in their AttributeListImpl class.

In our AttributeNode class, we have the following fields: "*name*" of type String; "*value*" of type String; "*fixed*" of type boolean; "*fixedCount*" of type int; and the pointer to the next node, "*next*" of type MyAttributeNode. Since all the attributes are in the start tag, almost all the work on the attribute list is done on the event of start-element. In the event of start-element, the Attribute Engine searches for the headerName to see if it is already in the header list. If it is not in the list, it inserts the attributes into the Required section, each node having "*fixed*" field as true, and "*fixedCoun*t" as 1. This is the only time the engine inserts into the Required section. Later some nodes may be deleted or moved from the Required section into Implied section, but no new attribute will be added to the Required section. If the header already exists in the header list, we then insert the attribute into the NewComer section waiting to be processed, or partitioned into the two sections, Required and Implied. The partition not only involves the attributes in the

NewComer section, but also other two sections because each time on another occurrence of an element we have to check if the attribute in the Required section still qualify for #REQUIRED and the fixed attributes still qualify for #FIXED. (We do not have a separate section for Fixed but we have a boolean field “fixed” in each node of AttributeNode). This is referred as the “juggling”.

Juggling is done as follows: First check if the attributes in the required section still qualify for #REQUIRED and if the fixed still qualify for #FIXED. To do this, for each attribute in the Required section, check if it is also in the NewComer list. If yes, check if the “fixed” field is still true. If it is still true, then check if the attribute value is preserved. If yes, the value is preserved, then increment the fixedCount by calling incrementFixedCount() method. If the attribute value is not preserved, set fixed to be false. Then remove the attribute from the NewComer section no matter the attribute value is preserved or not. If the attribute in the Required section is not found in the NewComer section, this means this attribute no longer qualify for #REQUIRED. We add it to Implied section and remove it from the Required section.

After this, we insert the rest of the attribute in the NewComer section to the Implied section. For each attribute in the NewComer section, check if it is in the Implied section. If it is already there, then do nothing. If it is not in there, insert it into the Implied section. And finally we clear the NewComer section for later use.

When we output the attribute list declarations of the DTD, we first output the Required section. We check if the “fixed” field is true and fixedCount is greater than the preset threshold. If yes, then output as “#FIXED” followed by the attribute value. If not, then output as “#REQUIRED”. And then output the Implied section as “#IMPLIED”.

The juggling and the output algorithms are shown in Figure 9.

```
1. On start-element event, search for the headerName to see if it is already in the header list.

   11. if not,
       Insert the attributes into the Required section, each node having "fixed" field as true,
       and "fixedCount" as 1.

   12. if yes,
       121. insert the attributes into the NewComer section.
       122. check if the attributes in the Required section still qualify for
            required and if fixed still qualify for fixed.
            1221. for each attribute in the Required section, check if it is
                  also in the NewComer section
                  12211. if yes
                         check if the "fixed" field is still true
                          122111. if yes,  check if attr-value is preserved
                                  1221111 yes, incrementFixedCount
                                  1221112 no, set fixed=false
                                122112  (no matter yes or no) remove attr from NewComer
                        12212  if not,
                               add it to Implied section
                               remove it from Required section
                  1222. insert the rest of attr in NewComer into Implied section, w/o redudancy
                        12221 for each attr in NewComer, check if it is in Implied section
                              122211 if yes, do nothing
                              122212 if not, insert it into #IMPLIED
                        12222 clear NewComer for later use

2. When output Attributes Declarations

   21. output Required section
       check fixed field, if true, and fixedCount>=fixedCountThreshod
       if true, output "#FIXED"  and then the attr-value
       if not, output "#REQUIED"
   22. output Implied section with "#IMPLIED"
```

Figure 9. Algorithms for the Attribute Engine in Pseudo Code

6.4.3 Reduction Engine

We now describe to the Reduction Engine. In the event of end-document, the DTD is already built. Before we output the DTD, we want to reduce it to a simpler and more reasonable form. There are equivalent reductions and relaxing reductions as discussed in Chapter 5. The Reduction Engine has two parts, factorization and degeneration. The factorization is the unique feature of our DTD Inference Engine. It greatly simplifies the output DTD in many instances. Let us look at an example. Suppose we have an element E, with the children sequence AX | BY | CZ | AY | CX | BZ | CY | AZ | BX. Many existing DTD generators leave this string as is without any additional simplification. Michael Key's generator, for example, does the lazy collapsing. It will collapse this into the non-ordered degenerate form (A | B | C | X | Y | Z)*.

As we can see, this reduction results in the loss of information of the original internal structure. Instead, our engine applies a factorization technique. This is very similar to the polynomial factorization. The analogy here is that the sequence or concatenation of the children is analogous to the polynomial multiplication. The vertical bar (OR) is analogous to polynomial addition. Each section separated by the vertical bars is analogous to one term in a multi-variable polynomial. The difference between the two is that in polynomial, the order of the factors in each term does not matter, while it does in the case of the child sequences. When performing our factorization, we pay respect to the order. We first do a left factorization followed by a right factorization. After the left factorization, the sequences in the above example becomes

A, (X|Y|Z) | B, (Y|Z|X) | C, (Z|X|Y).

Please note that the order of the sections separated by the vertical bars does not matter, meaning X|Y|Z, Y|Z|X, and Z|X|Y are all the same. There is still a right common factor, which is (X|Y|Z). In order to recognize this common factor, we first sort all the sections according to their lexicographical order before we start factoring. Finally the output of our engine for this example is

<!ELEMENT E ((A|B|C), (X|Y|Z))>.

It indicates that the element E has two children. The first is selected from A, B, or C and the second is selected from X, Y, or Z. We get much better information about the structure of the element than either the lazy concatenation of nine terms or the lazy collapsing, which make even XXZCBYA a possible child sequence of element E.

6.5 Handling Multiple XML Documents with the File Handler

There are many occasions on which we have multiple documents conforming to one DTD but the DTD is missing. A constraint is, however, that all the documents have to have the same root name. We are trying to infer the DTD information from these documents. Generally speaking, more instances of documents provide us with more information about the missing DTD than just a single document. However, there are also some difficulties that need to be addressed.

Most importantly, the parser once can only parse one document and the document has to have a tree structure. If we concatenate all the documents together, then the structure is no longer a tree, but a forest. In that case, the parser will throw an exception. If we start the parser on each document at a time and start the parser multiple times, each

time the parser and the DTD engine build a DTD for each document. It is a difficult task to merge these DTDs into one coherent DTD.

Our approach is to create a new document, the super document. We call the root of the super document SuperRoot. And we make the SuperRoot the parent of the roots of all the input documents. Doing so, we arrive at just a single tree. The parser can be invoked just once on this super tree and the DTD Inference Engine can gather information from all the documents.

The FileHandler doesn't have to physically concatenate all the document files. Actually what it does is to create a new document with the root name SuperRoot and then use external entity reference to link all the documents into this super document.

Before parsing the super document, the File Handler does a check on the root names of all the documents. If it finds that any of the documents has a different root name, it will throw an exception. We then know that these documents cannot be possibly derived from the same DTD. Another task of the File Handler is to strip off the XML headers like <?xml version="1.0"?>, which may appear on top of each document. While this is OK in the beginning of the document, XML headers cannot occur at any other location. What was on top of each document now is in the middle of the super document and XML does not like that. After stripping the headers of each file, the File Handler writes it to a new temporary file for each file. After the handling by the File Handler, the parser and the DTD Inference Engine will work on the super document. At the end when all work is done, the File Handler cleans the temporary files.

6.6 Complexity of the DTD Inference Engine

In order to get a feel for the efficiency of our DTD Inference Engine, we provide a brief, informal analysis of its run-time behavior as a function of the size of the input document. Let *n* be the number of elements in the document. We use *n* as the instance characteristic for the ensuing complexity analysis.

6.6.1 Number of Nodes in the DTD

We first need to find out the number of nodes in the DTD three dimensional linked list data structure. We first consider only the element declaration part of the DTD and leave the attribute declaration part for a later discussion. With a little observation we find out that each element, except for the root element, appears twice in the DTD, once as an entry heading in the parent list, the second time as the child in the children list of its parent. In the worst case, when all the elements are distinct, we have $2n$-1 nodes in the DTD. If some elements have more than one occurrence in the document, the DTD may be smaller. In practice, there are a lot of repetitions of the elements in the document. As a result, the size of the DTD is much smaller than the source document. That justifies that the DTD is a structural summary of the document.

The distribution of these nodes among the element lists may be quite different because the structure of the document tree may vary dramatically. One extreme is that the tree is a chain with a single element in each level and all the elements are distinct. In this case the DTD has exactly *n* entries with single child in the children list for each entry. The other extreme is that the tree is a star. There are only two levels. All the elements except the root are in the second level and are the children of the root. We see the number of children of an element could be as large as $O(n)$.

However, if we assume that all the trees have a constant degree, which does not grow with the document size n, we can simplify the analysis a little bit. In fact, this is a reasonable assumption. In practice, like in e-commerce applications, we hardly encounter a document with a degree greater than twenty.

6.6.2 Time Complexity of the Element Engine

If we assume a constant degree of the XML document trees, we know the length of each section is no longer than the degree of the tree, which is a constant. However, the number of sections contained in one element could still be as high as O(n) because one element may occur in the document many times. We can make the bound a little tighter. Let us assume that we have k number of elements each with O(n) number of sections. We claim that k must be O(1). Otherwise, if k is greater than O(1), the total number of sections, and hence the total number of nodes in the DTD, exceeds O(n), which contradicts the fact that the worst case total number of nodes in the DTD is 2n-1.

Let us summarize the picture of the DTD structure: the worst case number of entries is n. the worst case number of sections contained in one element is O(n), but the total number of this kind of large lists is O(1).

The DTD Inference Engine is based on a depth first traversal of the document tree. The depth first traversal takes O(n) time if the time spent at each node is constant. Now Let us find what is the time spent at each node. The push and pop operations of the stack take constant time per node. The append operation takes constant time per node because we maintain the lastSection and lastChild pointers. Subsequence check is more expensive. If one entry has O(n) sections, checkSubsequence may take O(n^2) time. And we know that this kind of entries does not exceed O(1). So the total time is O(n^2).

6.6.3 Time Complexity of the Attribute Engine

The complexity analysis of the Attribute Engine is simpler. It is reasonable to assume that the maximum number of attributes of each element doesn't grow with the document size n. For each element, appending the new attribute to the NewComer section takes constant time. The juggling of the attributes for each element also takes constant time because the size of the three sections of the attribute list, Required, Implied and NewComer are all constant. Hence the complexity of the Attribute Engine is O(n).

6.6.4 Time Complexity of the Reduction Engine

If the number of sections of an element is O(n), then the sort takes O(n^2) time. Factorization also takes O(n^2) time. As we have analyzed before, the number of such elements is O(1). So the total time is still O(n^2). Degeneration takes O(n) time. The total time for the Reduction Engine is O(n^2). We could have implemented a faster sorting algorithm using O($nlogn$) time. Our choice is based on the faith that in practice, we never have an element with O(n) sections.

All in all, the total time for the DTD Inference Engine is bounded by O(n^2).

CHAPTER 7
INCREMENTAL MAINTENANCE OF THE DTD

Although the practical complexity of the DTD Inference Engine is almost linear, there are some occasions when the complexity is close to O(n^2). In addition, there are situations when the source XML is dynamically changing and these changes occur often and fast. In those cases it may be difficult and inefficient to continue updating the inferred DTD at the same pace at which the source is changing. If we invoke the DTD Inference Engine on the document every time a change occurs in the source, DTD inference becomes an expensive operation. If the change is small, we can consider incremental maintenance of the DTD. That is, if the change is small, we do not infer the DTD from scratch. Instead, we make direct changes on the DTD according to the change in the source.

To do so, we first have to specify a complete set of editing operations on the source XML document. Chawathe et al. [42] studied change detection in hierarchically structured information and proposed a set of editing operations: node insert, node delete, node update and sub-tree move. Considering the XML as a special hierarchical structure and the nature of our DTD Inference Engine, we use the following set of editing operations:

insertLeaf, deleteLeaf, addAttribute and deleletAttribute.

The first two operations change the tree structure of the document as follows: InsertLeaf inserts a leafNode in the document tree. DeletLeaf deletes a leaf node from the

document tree. The other two operations addAttribute and deleteAttribute only change the attributes of an element.

To use a set of editing operations to describe changes, the set has to be complete. That is, starting from any document, applying an sequence of primitive editing operations, we should be able to arrive at any destination document. Besides completeness, we may add derived editing operations into this set for convenience.

The set of editing operations is complete as we can see by deleting the leaf node one by one. We can delete the entire tree and by inserting the leaf node one by one we can build any tree. So deleting leaf nodes and inserting leaf nodes enables us to change any tree into any other tree. Similarly, deleting attribute and adding attribute allows us to change any set of attributes to any other set of attributes.

We do not support other editing operations like delete sub-tree, move sub-tree, or update the name of an element. First of all, these operations can be derived from the four primitive operations we just proposed. Second, we use SAX, which is an event-based parser rather than a tree-based parser, and does not build an internal tree to represent the document. It just makes a one time traversal of the tree. After the traversal, the summary information of the structure is built into the DTD. However, the original tree structure is no longer kept in memory. To support those other operations, we would need to keep the information of the original tree.

The editing sequence for the original document is stored in a log file. In order to incrementally maintain the previously inferred DTD, the maintenance module of the DTD Inference Engine will read the log file, read the original DTD, and then apply the changes directly.

We use the following structure for the editing sequences in the log:

insert-leaf	parentName	leafName		
delete-leaf	parentName	leafName		
add-attribute	elementName	attributeName	attributeType	attributeValue
delete-attribute	elementName	attributeName		

The format is self-explanatory. The first token specifies the type of the operation while the rest of the tokens are the parameters. To insert a leaf node, we need to specify the parent name and the name of the new node, which will become a new leaf node. To delete a node, we also have to specify the parent name and the leaf name. To add an attribute, we have to specify the element name, into which we want to add the attribute, as well as the attribute name, attribute type, and the attribute value. To delete an attribute from an element, we only have to specify the element name and the attribute name.

Figure 10 shows an example of a log file and the documents as well as DTDs before and after the changes. A leaf node “make” in inserted under “vehicle” and a leaf node “model” is inserted under the same “vehicle”. A leaf node PCDATA “Toyota” is added under “make” and a leaf node PCDATA “Corolla” is added under “model”. Attribute “color” with value “white” is added to “vehicle”. Leaf nodes “Ford” and “Taurus” are deleted from their parents “make” and “model” respectively. As we can see the changes made in the DTD, the order in “make” and “model” in “vehicle” is degeneralized. The attribute list “color” is added to the element “vehicle”.

We now state our policies on the DTD maintenance.

Original XML Document:

```
<?xml version="1.0">
<vehicles>
        <vehicle>
                <make>Ford</make>
                <model>Taurus</model>
        <vehicle>
                <make>Mazda</make>
                <model>626</model>
</vehicles>
```

Original DTD:

```
<!DOCTYPE vehicles[
<!ELEMENT vehicles (vehicle*)>
<!ELEMENT vehicle (make, model)>
<!ELEMENT make (#PCDATA)>
<!ELEMENT model (#PCDATA)>
]>
```

Log file:

```
insert-leaf     vehicle     make
insert-leaf     vehicle     model
insert-leaf     make        Toyota
insert-leaf     model       Corolla
delete-leaf     make        Ford
delete-leaf     model       Taurus
add-attribute   vehicle     Color       CDATA white
```

XML Document after the change:

```
<?xml version="1.0">
<vehicles>
        <vehicle color = "while">
                <make>Toyota</make>
                <model>Corolla</model>
        <vehicle>
                <make>Mazda</make>
                <model>626</model>
</vehicles>
```

DTD after the change:

```
<!DOCTYPE vehicles[
<!ELEMENT vehicles (vehicle*)>
<!ELEMENT vehicle (make, model)*>
<!ATTLIST vehicle color CDATA #IMPLIED>
<!ELEMENT make (#PCDATA)>
<!ELEMENT model (#PCDATA)>
]>
```

Figure 10. A Sample Log and the Changes to the DTD

7.1 Insert Leaf

On insert-leaf, we first degenerate the parent element and then we add the leaf name to the child list of the parent. Next, we check if the name of the leaf is already declared in the element list. If it is not declared, we append an entry <!ELEMENT leafname (#PCDATA)>.

If it is already in the element list, we check if it has #PCDATA as its child. If not, we add #PCDATA.

7.2 Delete Leaf

On delete-leaf, we just simply degenerate the parent elementHeader. In both cases, we apply the degeneration to the parent header. This is also because the fact that we do not keep the internal tree of the original document. We do not have detailed information about the structure of the original document in the DTD. Doing so, it loses some information because of the degeneration but at least we still can obtain a sound DTD for the updated document. The new document conforms to the modified DTD even if we lose some detailed information. The user has the option to trade accuracy for speed, or start the DTD Inference Engine all over again, which takes time but increases accuracy. However, if the header has only one section, no information is lost in the degeneration. One analogy is the JPEG format to store images. You scan in an image in JPEG format. Each time you save the file as JPEG format after editing, you lose information. The image gets fuzzier each time, but the user decides if it is acceptable or not.

For the deleting leaf operation, we only do a degeneration of the parent header but we do not delete anything from the DTD. This is because the wild card '*' in the degenerate form. Even we do not delete anything from the DTD, the DTD is still sound with respect to the new document.

7.3 Add Attribute and Delete Attribute

When we add an attribute to an element, we just want to put this attribute to the Implied section. We know the newly added attribute cannot previously be in the Required section, because, if it were #REQUIRED, then it appears in every occurrence of the element and we can't add an attribute twice with the same name. If this element previously didn't have this attribute at all and this element has multiple occurrences, because we only add this attribute to one occurrence, it certainly does not quality for #REQUIRED. We should put it in Implied section. If this element has only one occurrence and we add a new attribute, it qualifies for #REQUIRED according to our previous stated rules (Chapter 6). However, adding it as an #IMPLIED attribute is acceptable because this is a more general form and the resulting DTD is still sound.

When we delete an attribute, we only check if it was previously in the Required section. If it was, we remove it and add it to the Implied section.

CHAPTER 8
CONCLUSION

The DTD Inference Engine is an important component of the I-Wiz project. The study of DTD inference also has its importance in the field of context-free languages. In this thesis, we motivate the need for DTD inference. We start with the XML specifications and conduct theoretical research of DTD inference in the context of context-free languages. We define the concept of Kernel Derivation Tree, the sound, the tight and the closure DTD. We investigate the relationship of multiple derivation trees from a given grammar and multiple grammars for a given derivation tree. We prove two theorems and reach the conclusion that a finite language has a finite number of KDTs while an infinite language has infinite number of KDTs. The tight DTD or the closure DTD may not exit for a given set of XML documents with the same root element. We study the reduction of DTD and make the classification of reductions as equivalent reduction and relaxing reduction.

We describe the design and implementation of the DTD Inference Engine. We first state the policies and rules we adopt for the element declarations and attribute list declarations. We then design the three-dimensional linked list data structure to represent the DTD. We design the architecture of the DTD Inference Engine with the Element Engine, Attribute Engine and Reduction Engine as components. We describe the algorithms and analyze the complexity. We find our solution for the multiple documents

handling mechanism by creating a super XML document with the root element SuperRoot. We finally discuss the incremental maintenance of the DTD.

8.1 Result and Verification

Besides the module tests, we tested our DTD Inference Engine with different sources of XML data including the sample XML documents in the book "The XML Bible" by E. R. Harold [27]. The most important test documents are XML representations of the periodical elements table, excerpts from Shakespear's works and from the Bible. We also tested our DIE on the XML documents in the e-commerce domain found on the Commerce One, Inc.'s Web site as well as the XML version of the current and past issues of SIGMOD Record. We compared our inferred DTD with the original DTD and with the DTD inferred by other DTD generators, such as Michael Kay's and Fred. In all the cases, our inferred DTD are correct and on many instances superior to those inferred by other DTD generators. To demonstrate, we give one example from Commerce One Web site. In Appendix B, we list the rest of the XML documents found on Commerce One Web site and compare the original DTDs with the inferred DTDs generated by our DTD Inference Engine. We don't list the periodical table, Shakespear's works and the Bible because they are too long. The example we are going to analyze is the Invoice.xml document from Commerce One (See Appendix B).

We notice in the document, the element **BaseItemDetail** has two occurrences. In the first Occurrence, it has children **LineItemNum, SupplierPartNum, Quantity** and in the second occurrence, it has children **LineItemNum, SupplierPartNum,**

ItemDescription, Quantity. The element declaration for **BaseItemDetail** inferred by our DTD Inference Engine is

<!ELEMENT BaseItemDetail (LineItemNum, SupplierPartNum, ItemDescription*, Quantity)>, which is correct and just summarises the structural information. (We have discussed the reason we use "*" instead of "?" in Chapter 6.) We also tested the same XML document with Michael Kay's DTD Generator. While for most other elements, Michael Kay's DTD Generator gives the same result as that by our DTD Inference Engine, for the element **BaseItemDetail**, it generates the element declaration as <!ELEMENT BaseItemDetail (LineItemNum, SupplierPartNum, ItemDescription, Quantity)*>, which is the degenerate form and it is too general to capture the structural information of element **BaseItemDetail** in the document.

8.2 Contributions

Our direct contribution to the I-Wiz project is the design and implementation of the DTD Inference Engine, which interfaces the DRE engine in the I-Wiz project. We designed the three-dimensional linked list data structure to represent the DTD and to accelerate the engine.

The DTD inference is also an active research area outside the I-Wiz project, in the general language background. We contributed to the theoretical study of DTD inference by defining and clarifying some key concepts like Kernel Derivation Tree, sound DTD, tight DTD and closure DTD. We gave two theorems revealing the relationship between the number of KDTs of a single grammar and the number of grammars to a single KDT. We revealed the non-existence of the closure DTD on certain occasions.

Our implemented DTD Inference Engine has three major enhanced features, namely the factorization reduction, the multiple documents handling ability and the incremental maintenance.

We believe our DTD Inference Engine gives more insight in the theoretical study of DTD inference and our implementation of the DTD Inference Engine will benefit many XML applications not limited to the I-Wiz project.

8.3 Future Work

We would like to point out that this implementation of the DTD Inference Engine is not the end point of the research. We indicate several possible directions for extending the research described here.

First, on the incremental maintenance, we can try to explore the automatic detection of the changes. However, automatic detection of change for a hierarchical structure could be expensive itself. The engine should be smart enough to make an decision on its own that when is better to detect the change and when is better to restart the DTD Inference Engine.

Second, with the increasing support of XML Schema, we can explore the Schema inference for XML documents. Schema is more powerful and there are more problems in the Schema inference.

In conclusion, DTD inference is a very interesting and fast growing research area. We believe we will see many interesting new approaches in both theoretical study and implementations in the near future.

APPENDIX A
FORMAL XML SPECIFICATION PERTAINING TO DTD IN EBNF FORM

Document Type Definition

```
[28]  doctypedecl ::=   '<!DOCTYPE' S Name (S ExternalID)? S? ('['
(markupdecl | PEReference | S)* ']' S?)? '>'    [     VC: Root Element
Type ]
[29]  markupdecl  ::=   elementdecl | AttlistDecl | EntityDecl |
NotationDecl | PI | Comment   [     VC: Proper Declaration/PE Nesting ]
                        [     WFC: PEs in Internal Subset ]
```

Element Type Declaration

```
[45]   elementdecl ::=   '<!ELEMENT' S Name S contentspec S? '>'
       [ VC: Unique Element Type Declaration ]
[46]   contentspec ::=   'EMPTY' | 'ANY' | Mixed | children
```

Element-content Models

```
[47]   children    ::=   (choice | seq) ('?' | '*' | '+')?
[48]   cp    ::=   (Name | choice | seq) ('?' | '*' | '+')?
[49]   choice      ::=   '(' S? cp ( S? '|' S? cp )* S? ')'  [     VC:
Proper Group/PE Nesting ]
[50]   seq   ::=   '(' S? cp ( S? ',' S? cp )* S? ')'  [     VC: Proper
Group/PE Nesting ]
```

Mixed-content Declaration

```
[51]   Mixed ::=   '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*'
                   | '(' S? '#PCDATA' S? ')'     [     VC: Proper
Group/PE Nesting ]
                        [     VC: No Duplicate Types ]
```

Attribute-list Declaration

```
[52]   AttlistDecl ::=   '<!ATTLIST' S Name AttDef* S? '>'
[53]   AttDef      ::=   S Name S AttType S DefaultDecl
```

Attribute Types

```
[54]   AttType     ::=   StringType | TokenizedType | EnumeratedType
[55]   StringType  ::=   'CDATA'
[56]   TokenizedType     ::=   'ID'  [     VC: ID ]
                        [     VC: One ID per Element Type ]
                        [     VC: ID Attribute Default ]
```

```
                    | 'IDREF'   [     VC: IDREF ]
                    | 'IDREFS'  [     VC: IDREF ]
                    | 'ENTITY'  [     VC: Entity Name ]
                    | 'ENTITIES'      [     VC: Entity Name ]
                    | 'NMTOKEN' [     VC: Name Token ]
                    | 'NMTOKENS'      [     VC: Name Token ]
```

Enumerated Attribute Types

```
[57]   EnumeratedType    ::=   NotationType | Enumeration
[58]   NotationType      ::=   'NOTATION' S '(' S? Name (S? '|' S?
Name)* S? ')'     [     VC: Notation Attributes ]
[59]   Enumeration ::=   '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'
       [     VC: Enumeration ]
```

Attribute Defaults

```
[60]   DefaultDecl ::=   '#REQUIRED' | '#IMPLIED'
                    | (('#FIXED' S)? AttValue)    [     VC: Required
Attribute ]
                          [     VC: Attribute Default Legal ]
                          [     WFC: No < in Attribute Values ]
                          [     VC: Fixed Attribute Default ]
```

Entity Declaration

```
[70]   EntityDecl  ::=   GEDecl | PEDecl
[71]   GEDecl      ::=   '<!ENTITY' S Name S EntityDef S? '>'
[72]   PEDecl      ::=   '<!ENTITY' S '%' S Name S PEDef S? '>'
[73]   EntityDef   ::=   EntityValue | (ExternalID NDataDecl?)
[74]   PEDef ::=   EntityValue | ExternalID
```

External Entity Declaration

```
[75]   ExternalID  ::=   'SYSTEM' S SystemLiteral
                    | 'PUBLIC' S PubidLiteral S SystemLiteral
[76]   NDataDecl   ::=   S 'NDATA' S Name  [     VC: Notation Declared ]
```

Text Declaration

```
[77]   TextDecl    ::=   '<?xml' VersionInfo? EncodingDecl S? '?>'
```

Encoding Declaration

```
[80]   EncodingDecl      ::=   S 'encoding' Eq ('"' EncName '"' | "'"
EncName "'" )
[81]   EncName     ::=   [A-Za-z] ([A-Za-z0-9._] | '-')*     /*
       Encoding name contains only Latin characters */
```

Notation Declarations

[82] NotationDecl ::= '<!NOTATION' S Name S (ExternalID | PublicID) S? '>'

[83] PublicID ::= 'PUBLIC' S PubidLiteral

APPENDIX B
OUTPUT DTDS OF THE DIE FOR COMMERCE ONE E-COMMERCE
APPICATION XML DOCUMENTS

DOCUMENT: Invoice.xml

```
<?soxtype urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0?>

<!-- invoice1.xml is an example of a simple invoice for 10 sets of -->
<!-- break pads 12 cases of 20-50 motor oil -->
<!-- all the fields in this invoice are required by Invoice.sox -->

<Invoice>
        <!-- InvoiceHeader contains general information about that applies -->
        <!-- to the entire invoice -->
        <InvoiceHeader>
                <InvoiceDate>19990517</InvoiceDate> <!-- May 17th, 1999 -->

                <ContractNumber>ABC124</ContractNumber>
                <PriceListNumber> 5 </PriceListNumber>
                <PriceListVersionNumber>1.2</PriceListVersionNumber>

                <BuyersCatalogNumber>56</BuyersCatalogNumber>

                <!-- this number was generated by the Suppliers systems -->
                <SupplierOrderNumber>az152</SupplierOrderNumber>

                <!-- BuyerOrderNumber is a number generated by the Buyer -->
                <!-- it is the Buyers Purchase Order number -->
                <BuyerOrderNumber> 12_df_1567 </BuyerOrderNumber>

                <!-- Currency is not normally in InvoiceHeader -->
                <!-- the invoice is always in a single currency -->
                <InvoiceCurrency>USD</InvoiceCurrency>
        </InvoiceHeader>

        <!-- InvoiceParties contains names and address of parties and -->
        <!-- their functions -->
        <InvoiceParties>
                <Buyer>
                        <NameAddress>
                                <Name1>Ralph`s Automotive Parts</Name1>
                                <Address1>10 Main St.</Address1>
                                <City>Boulder Creek</City>
                                <StateOrProvince>California</StateOrProvince>
                                <PostalCode>96005</PostalCode>
                                <Country>US</Country>
```

```
                </NameAddress>
        </Buyer>
        <Supplier>
                <NameAddress>
                        <Name1>ABC Wholesale</Name1>
                        <Address1>1222 Industrial Park way</Address1>
                        <City>South San Francisco</City>
                        <StateOrProvince>California</StateOrProvince>
                        <PostalCode>96045</PostalCode>
                        <Country>US</Country>
                </NameAddress>
        </Supplier>
</InvoiceParties>

<!-- ListOfInvoiceDetail has the actual line items -->
<ListOfInvoiceDetail>
        <!-- this is the first line. It is for 10 sets of break pads -->
        <InvoiceDetail>
                <BaseItemDetail>
                        <!-- The orginal line number in the purchase -->
                        <!-- order was 1 -->
                        <LineItemNum>1</LineItemNum>
                        <SupplierPartNum>
                                <PartNum>
                                        <Agency AgencyID="AssignedBySupplier"/>
                                        <PartID>SKU123</PartID>
                                </PartNum>
                        </SupplierPartNum>
                        <Quantity>
                                <Qty>10</Qty>
                                <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
                        </Quantity>
                </BaseItemDetail>
                <InvoiceUnitPrice>13.95</InvoiceUnitPrice>

        </InvoiceDetail>

        <!-- this is the second line. It is for 12 cases of -->
        <!-- 20-50 motor oil. -->
        <InvoiceDetail>
                <BaseItemDetail>
                        <!-- The orginal line number in the purchase -->
                        <!-- order was 10 -->
                        <LineItemNum>10</LineItemNum>
                        <SupplierPartNum>
                                <PartNum>
                                        <Agency AgencyID="AssignedBySupplier"/>
                                        <PartID>SKUABC</PartID>
                                </PartNum>
                        </SupplierPartNum>
                        <ItemDescription>
                                12 cases of motor oil. each case contains 24, 1
                                quart bottles
                        </ItemDescription>
                        <Quantity>
                                <Qty>12</Qty>
```

```
                                        <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
                                    </Quantity>
                                </BaseItemDetail>
                                <InvoiceUnitPrice>15.75</InvoiceUnitPrice>
                        </InvoiceDetail>
                </ListOfInvoiceDetail>

                <InvoiceSummary>
                        <SubTotal>328.50</SubTotal>

                        <Tax>
                                <TaxPercent>8.2</TaxPercent>
                                <Location>Santa Cruz County</Location>
                                <TaxAmount>26.947</TaxAmount>
                                <TaxableAmount>328.50</TaxableAmount>
                        </Tax>

                        <Total>355.437</Total>
                </InvoiceSummary>
</Invoice>
```

DTD: Invoice.dtd

```
<!DOCTYPE Invoice [

<!ELEMENT Invoice (InvoiceHeader, InvoiceParties, ListOfInvoiceDetail, InvoiceSummary)>

<!ELEMENT InvoiceHeader (InvoiceDate, ContractNumber, PriceListNumber, PriceListVersionNumber,
BuyersCatalogNumber, SupplierOrderNumber, BuyerOrderNumber, InvoiceCurrency)>

<!ELEMENT InvoiceDate (#PCDATA)>

<!ELEMENT ContractNumber (#PCDATA)>

<!ELEMENT PriceListNumber (#PCDATA*)>

<!ELEMENT PriceListVersionNumber (#PCDATA)>

<!ELEMENT BuyersCatalogNumber (#PCDATA)>

<!ELEMENT SupplierOrderNumber (#PCDATA)>

<!ELEMENT BuyerOrderNumber (#PCDATA*)>

<!ELEMENT InvoiceCurrency (#PCDATA)>

<!ELEMENT InvoiceParties (Buyer, Supplier)>

<!ELEMENT Buyer (NameAddress)>

<!ELEMENT NameAddress (Name1, Address1, City, StateOrProvince, PostalCode, Country)>
```

<!ELEMENT Name1 (#PCDATA)>

<!ELEMENT Address1 (#PCDATA)>

<!ELEMENT City (#PCDATA)>

<!ELEMENT StateOrProvince (#PCDATA)>

<!ELEMENT PostalCode (#PCDATA)>

<!ELEMENT Country (#PCDATA)>

<!ELEMENT Supplier (NameAddress)>

<!ELEMENT ListOfInvoiceDetail (InvoiceDetail*)>

<!ELEMENT InvoiceDetail (BaseItemDetail, InvoiceUnitPrice)>

<!ELEMENT BaseItemDetail (LineItemNum, SupplierPartNum, ItemDescription*, Quantity)>

<!ELEMENT LineItemNum (#PCDATA)>

<!ELEMENT SupplierPartNum (PartNum)>

<!ELEMENT PartNum (Agency, PartID)>

<!ELEMENT Agency EMPTY>
<!ATTLIST Agency AgencyID CDATA #REQUIRED>

<!ELEMENT PartID (#PCDATA)>

<!ELEMENT Quantity (Qty, UnitOfMeasure)>

<!ELEMENT Qty (#PCDATA)>

<!ELEMENT UnitOfMeasure (UOM)>

<!ELEMENT UOM (#PCDATA)>

<!ELEMENT InvoiceUnitPrice (#PCDATA)>

<!ELEMENT ItemDescription (#PCDATA*)>

<!ELEMENT InvoiceSummary (SubTotal, Tax, Total)>

<!ELEMENT SubTotal (#PCDATA)>

<!ELEMENT Tax (TaxPercent, Location, TaxAmount, TaxableAmount)>

<!ELEMENT TaxPercent (#PCDATA)>

<!ELEMENT Location (#PCDATA)>

<!ELEMENT TaxAmount (#PCDATA)>

<!ELEMENT TaxableAmount (#PCDATA)>

```
<!ELEMENT Total (#PCDATA)>

]>
```

DOCUMENT: AvailabilityCheckRequest.xml

```
<AvailabilityCheckRequest>
        <!-- The supplier of the PartKeys to be quoted-->
        <AvailabilityCheckHeader>
        <SupplierID>
                <Reference>
                        <RefNum>OD1233</RefNum>
                </Reference>
        </SupplierID>

        <!-- The buyer account code -->
        <AccountCode>
                <Reference>
                        <RefNum>OD11222S</RefNum>
                </Reference>
        </AccountCode>
        </AvailabilityCheckHeader>

        <!-- A list of order items:  PartKey, quote date, quantity-->
        <!-- The ordering of items returned is guaranteed to match the ordering -->
        <!-- of items in the AvailabilityCheckRequest. -->
        <ListOfBaseItemDetail>
        <BaseItemDetail>
                <LineItemNum>1</LineItemNum>
                <SupplierPartNum>
                        <PartNum>
                                <Agency AgencyID="AssignedBySupplier" />
                                <PartID>PK122122</PartID>
                        </PartNum>
                </SupplierPartNum>
                <Quantity>
                        <Qty>10</Qty>
                        <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
                </Quantity>
        </BaseItemDetail>

        <BaseItemDetail>
                <LineItemNum>2</LineItemNum>
                <SupplierPartNum>
                        <PartNum>
                                <Agency AgencyID="AssignedBySupplier" />
                                <PartID>PK122122</PartID>
                        </PartNum>
                </SupplierPartNum>
                <Quantity>
                        <Qty>1</Qty>
                        <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
                </Quantity>
        </BaseItemDetail>
```

```
        </ListOfBaseItemDetail>
</AvailabilityCheckRequest>
```

DTD: AvailabilityCheckRequest.dtd

```
<!DOCTYPE AvailabilityCheckRequest [
<!ELEMENT AvailabilityCheckRequest (AvailabilityCheckHeader, ListOfBaseItemDetail)>
<!ELEMENT AvailabilityCheckHeader (SupplierID, AccountCode)>
<!ELEMENT SupplierID (Reference)>
<!ELEMENT Reference (RefNum)>
<!ELEMENT RefNum (#PCDATA)>
<!ELEMENT AccountCode (Reference)>
<!ELEMENT ListOfBaseItemDetail (BaseItemDetail*)>
<!ELEMENT BaseItemDetail (LineItemNum, SupplierPartNum, Quantity)>
<!ELEMENT LineItemNum (#PCDATA)>
<!ELEMENT SupplierPartNum (PartNum)>
<!ELEMENT PartNum (Agency, PartID)>
<!ELEMENT Agency EMPTY>
<!ATTLIST Agency AgencyID CDATA #REQUIRED>
<!ELEMENT PartID (#PCDATA)>
<!ELEMENT Quantity (Qty, UnitOfMeasure)>
<!ELEMENT Qty (#PCDATA)>
<!ELEMENT UnitOfMeasure (UOM)>
<!ELEMENT UOM (#PCDATA)>
]>
```

DOCUMENT: AvailabilityCheckResult.xml

```
<?soxtype urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0?>
<!-- Instance of AvailabilityCheckResult -->
```

```
<AvailabilityCheckResult>
        <!-- The supplier of the PartKeys to be quoted-->
        <AvailabilityCheckHeader>
        <SupplierID>
                <Reference>
                        <RefNum>OD1233</RefNum>
                </Reference>
        </SupplierID>

        <!-- The buyer account code -->
        <AccountCode>
                <Reference>
                        <RefNum>OD11222S</RefNum>
                </Reference>
        </AccountCode>
        </AvailabilityCheckHeader>


        <ListOfAvailabilityResultItem>
                <AvailabilityResultItem>
                        <QuotedItem>
                                <BaseItemDetail>
                                        <LineItemNum>1</LineItemNum>
                                        <SupplierPartNum>
                                                <PartNum>
                                                        <Agency AgencyID="AssignedBySupplier"
/>
                                                        <PartID>PK122122</PartID>
                                                </PartNum>
                                        </SupplierPartNum>
                                        <Quantity>
                                                <Qty>10</Qty>

        <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
                                        </Quantity>
                                </BaseItemDetail>
                        </QuotedItem>
                        <AvailableQuantity>
                                <Quantity>
                                        <Qty>10</Qty>
                                        <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
                                </Quantity>
                        </AvailableQuantity>
                </AvailabilityResultItem>

                <AvailabilityResultItem>
                        <QuotedItem>
                                <BaseItemDetail>
                                        <LineItemNum>2</LineItemNum>
                                        <SupplierPartNum>
                                                <PartNum>
                                                        <Agency AgencyID="AssignedBySupplier"
/>
                                                        <PartID>PK122122</PartID>
                                                </PartNum>
```

```
                                        </SupplierPartNum>
                                        <Quantity>
                                                <Qty>1</Qty>

        <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
                                        </Quantity>
                                </BaseItemDetail>
                        </QuotedItem>
                        <AvailableQuantity>
                                <Quantity>
                                        <Qty>1</Qty>
                                        <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
                                </Quantity>
                        </AvailableQuantity>
                </AvailabilityResultItem>
        </ListOfAvailabilityResultItem>

        <AvailabilityCheckSummary>
        <AvailabilityItemErrors>0</AvailabilityItemErrors>
        </AvailabilityCheckSummary>

</AvailabilityCheckResult>
```

DTD: AvailabilityCheckResult.dtd

```
<!DOCTYPE AvailabilityCheckResult [

<!ELEMENT AvailabilityCheckResult (AvailabilityCheckHeader, ListOfAvailabilityResultItem,
AvailabilityCheckSummary)>

<!ELEMENT AvailabilityCheckHeader (SupplierID, AccountCode)>

<!ELEMENT SupplierID (Reference)>

<!ELEMENT Reference (RefNum)>

<!ELEMENT RefNum (#PCDATA)>

<!ELEMENT AccountCode (Reference)>

<!ELEMENT ListOfAvailabilityResultItem (AvailabilityResultItem*)>

<!ELEMENT AvailabilityResultItem (QuotedItem, AvailableQuantity)>

<!ELEMENT QuotedItem (BaseItemDetail)>

<!ELEMENT BaseItemDetail (LineItemNum, SupplierPartNum, Quantity)>

<!ELEMENT LineItemNum (#PCDATA)>

<!ELEMENT SupplierPartNum (PartNum)>

<!ELEMENT PartNum (Agency, PartID)>

<!ELEMENT Agency EMPTY>
```

```
<!ATTLIST Agency AgencyID CDATA #REQUIRED>

<!ELEMENT PartID (#PCDATA)>

<!ELEMENT Quantity (Qty, UnitOfMeasure)>

<!ELEMENT Qty (#PCDATA)>

<!ELEMENT UnitOfMeasure (UOM)>

<!ELEMENT UOM (#PCDATA)>

<!ELEMENT AvailableQuantity (Quantity)>

<!ELEMENT AvailabilityCheckSummary (AvailabilityItemErrors)>

<!ELEMENT AvailabilityItemErrors (#PCDATA)>

]>
```

DOCUMENT: OrderStatusRequest.xml

```
<!-- OrderStatusRequest1.xml is an example of an order status request document -->

<OrderStatusRequest>
        <OrderStatusHeader>
                <OrderStatusDate>19990809T01:01:01</OrderStatusDate>
                <OrderParty>
                        <BuyerParty>
                        <Party>
                        <NameAddress>
                                <Name1>Mr. Muljadi Sulistio</Name1>
                                <Name2>Attention: Business Service Division</Name2>
                                <Address1>1600 Riviera Ave</Address1>
                                <Address2>Suite# 200</Address2>
                                <City>Walnut Creek</City>
                                <StateOrProvince>CA</StateOrProvince>
                                <PostalCode>94596</PostalCode>
                                <Country>US</Country>
                        </NameAddress>
                        <OrderContact>
                                <Contact>
                                <ContactName>Mr. Mike Holloway</ContactName>
                                <Telephone>(925) 941-3333</Telephone>
                                <Email>mike.holloway@commerceone.com</Email>
                                <Fax>(925) 941-4555</Fax>
                                </Contact>
                        </OrderContact>
                        <ReceivingContact>
                                <Contact>
                                <ContactName>Mr. Debbie Dub</ContactName>
                                <Telephone>(925) 941-2222</Telephone>
                                <Email>debbie.dub@commerceone.com</Email>
                                <Fax>(925) 941-4555</Fax>
```

```
                    </Contact>
                </ReceivingContact>
                <ShippingContact>
                    <Contact>
                    <ContactName>Ms. John Wayne</ContactName>
                    <Telephone>(925) 941-1111</Telephone>
                    <Email>john.wayne@commerceone.com</Email>
                    <Fax>(925) 941-4555</Fax>
                    </Contact>
                </ShippingContact>
                </Party>
                </BuyerParty>
                <SupplierParty>
                <Party>
                <NameAddress>
                    <Name1>Millenium Supplier Corporation</Name1>
                    <Name2>Attention: Office Supply Division</Name2>
                    <Address1>355 Alameda Street</Address1>
                    <Address2>Suite 100</Address2>
                    <City>San Jose</City>
                    <StateOrProvince>CA</StateOrProvince>
                    <PostalCode>94588</PostalCode>
                    <Country>US</Country>
                </NameAddress>
                </Party>
                </SupplierParty>
            </OrderParty>
        </OrderStatusHeader>

        <ListOfOrderStatusDetailRequest>
        <OrderStatusDetailRequest>
            <OrderReference>
                <AccountCode>
                    <Reference>
                                <RefNum>OD11222S</RefNum>
                    </Reference>
                </AccountCode>
                <BuyerRefNum>
                    <Reference>
                                <RefNum>PO1221</RefNum>
                    </Reference>
                </BuyerRefNum>
                <SupplierRefNum>
                    <Reference>
                                <RefNum>009199111</RefNum>
                    </Reference>
                </SupplierRefNum>
            </OrderReference>
            <OrderDate>19990809T01:01:01</OrderDate>
        </OrderStatusDetailRequest>
        </ListOfOrderStatusDetailRequest>
</OrderStatusRequest>
```

DTD: OrderStatusRequest.dtd

<!DOCTYPE OrderStatusRequest [

<!ELEMENT OrderStatusRequest (OrderStatusHeader, ListOfOrderStatusDetailRequest)>

<!ELEMENT OrderStatusHeader (OrderStatusDate, OrderParty)>

<!ELEMENT OrderStatusDate (#PCDATA)>

<!ELEMENT OrderParty (BuyerParty, SupplierParty)>

<!ELEMENT BuyerParty (Party)>

<!ELEMENT Party (NameAddress, OrderContact*, ReceivingContact*, ShippingContact*)>

<!ELEMENT NameAddress (Name1, Name2, Address1, Address2, City, StateOrProvince, PostalCode, Country)>

<!ELEMENT Name1 (#PCDATA)>

<!ELEMENT Name2 (#PCDATA)>

<!ELEMENT Address1 (#PCDATA)>

<!ELEMENT Address2 (#PCDATA)>

<!ELEMENT City (#PCDATA)>

<!ELEMENT StateOrProvince (#PCDATA)>

<!ELEMENT PostalCode (#PCDATA)>

<!ELEMENT Country (#PCDATA)>

<!ELEMENT OrderContact (Contact)>

<!ELEMENT Contact (ContactName, Telephone, Email, Fax)>

<!ELEMENT ContactName (#PCDATA)>

<!ELEMENT Telephone (#PCDATA)>

<!ELEMENT Email (#PCDATA)>

<!ELEMENT Fax (#PCDATA)>

<!ELEMENT ReceivingContact (Contact)>

<!ELEMENT ShippingContact (Contact)>

<!ELEMENT SupplierParty (Party)>

<!ELEMENT ListOfOrderStatusDetailRequest (OrderStatusDetailRequest)>

<!ELEMENT OrderStatusDetailRequest (OrderReference, OrderDate)>

<!ELEMENT OrderReference (AccountCode, BuyerRefNum, SupplierRefNum)>

```
<!ELEMENT AccountCode (Reference)>

<!ELEMENT Reference (RefNum)>

<!ELEMENT RefNum (#PCDATA)>

<!ELEMENT BuyerRefNum (Reference)>

<!ELEMENT SupplierRefNum (Reference)>

<!ELEMENT OrderDate (#PCDATA)>

]>
```

DOCUMENT: OrderStatusResult.xml

```
<?soxtype urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0?>

<!-- OrderStatusResult1.xml is an example of an order status request document -->

<OrderStatusResult>
        <OrderStatusHeader>
                <OrderStatusDate>19990809T01:01:01</OrderStatusDate>
                <OrderParty>
                        <BuyerParty>
                        <Party>
                        <NameAddress>
                                <Name1>Mr. Muljadi Sulistio</Name1>
                                <Name2>Attention: Business Service Division</Name2>
                                <Address1>1600 Riviera Ave</Address1>
                                <Address2>Suite# 200</Address2>
                                <City>Walnut Creek</City>
                                <StateOrProvince>CA</StateOrProvince>
                                <PostalCode>94596</PostalCode>
                                <Country>US</Country>
                        </NameAddress>
                        <OrderContact>
                                <Contact>
                                <ContactName>Mr. Mike Holloway</ContactName>
                                <Telephone>(925) 941-3333</Telephone>
                                <Email>mike.holloway@commerceone.com</Email>
                                <Fax>(925) 941-4555</Fax>
                                </Contact>
                        </OrderContact>
                        <ReceivingContact>
                                <Contact>
                                <ContactName>Mr. Debbie Dub</ContactName>
                                <Telephone>(925) 941-2222</Telephone>
                                <Email>debbie.dub@commerceone.com</Email>
                                <Fax>(925) 941-4555</Fax>
                                </Contact>
                        </ReceivingContact>
                        <ShippingContact>
                                <Contact>
```

```
                        <ContactName>Ms. John Wayne</ContactName>
                        <Telephone>(925) 941-1111</Telephone>
                        <Email>john.wayne@commerceone.com</Email>
                        <Fax>(925) 941-4555</Fax>
                        </Contact>
                </ShippingContact>
                </Party>
                </BuyerParty>
                <SupplierParty>
                <Party>
                <NameAddress>
                        <Name1>Millenium Supplier Corporation</Name1>
                        <Name2>Attention: Office Supply Division</Name2>
                        <Address1>355 Alameda Street</Address1>
                        <Address2>Suite 100</Address2>
                        <City>San Jose</City>
                        <StateOrProvince>CA</StateOrProvince>
                        <PostalCode>94588</PostalCode>
                        <Country>US</Country>
                </NameAddress>
                </Party>
                </SupplierParty>
        </OrderParty>
</OrderStatusHeader>

<ListOfOrderStatusDetailResult>
<OrderStatusDetailResult>
        <OrderReference>
                <AccountCode>
                        <Reference>
                                        <RefNum>OD11222S</RefNum>
                        </Reference>
                </AccountCode>
                <BuyerRefNum>
                        <Reference>
                                        <RefNum>PO1221</RefNum>
                        </Reference>
                </BuyerRefNum>
                <SupplierRefNum>
                        <Reference>
                                        <RefNum>009199111</RefNum>
                        </Reference>
                </SupplierRefNum>
        </OrderReference>

        <OrderDate>19990809T01:01:01</OrderDate>
        <OrderStatusDate>19991001T01:01:01</OrderStatusDate>

        <Status>
                <StatusNote> Hello </StatusNote>
                <StatusEvent>
                        <StatusEventCodeElement>Processing</StatusEventCodeElement>
                </StatusEvent>
        </Status>

</OrderStatusDetailResult>
```

```
        </ListOfOrderStatusDetailResult>
      <OrderStatusCheckSummary><OrderStatusCheckItemErrors>0</OrderStatusCheckItemErrors></
OrderStatusCheckSummary>
</OrderStatusResult>
```

DTD: OrderStatusResult.dtd

```
<!DOCTYPE OrderStatusResult [

<!ELEMENT OrderStatusResult (OrderStatusHeader, ListOfOrderStatusDetailResult,
OrderStatusCheckSummary)>

<!ELEMENT OrderStatusHeader (OrderStatusDate, OrderParty)>

<!ELEMENT OrderStatusDate (#PCDATA)>

<!ELEMENT OrderParty (BuyerParty, SupplierParty)>

<!ELEMENT BuyerParty (Party)>

<!ELEMENT Party (NameAddress, OrderContact*, ReceivingContact*, ShippingContact*)>

<!ELEMENT NameAddress (Name1, Name2, Address1, Address2, City, StateOrProvince, PostalCode,
Country)>

<!ELEMENT Name1 (#PCDATA)>

<!ELEMENT Name2 (#PCDATA)>

<!ELEMENT Address1 (#PCDATA)>

<!ELEMENT Address2 (#PCDATA)>

<!ELEMENT City (#PCDATA)>

<!ELEMENT StateOrProvince (#PCDATA)>

<!ELEMENT PostalCode (#PCDATA)>

<!ELEMENT Country (#PCDATA)>

<!ELEMENT OrderContact (Contact)>

<!ELEMENT Contact (ContactName, Telephone, Email, Fax)>

<!ELEMENT ContactName (#PCDATA)>

<!ELEMENT Telephone (#PCDATA)>

<!ELEMENT Email (#PCDATA)>

<!ELEMENT Fax (#PCDATA)>

<!ELEMENT ReceivingContact (Contact)>
```

```
<!ELEMENT ShippingContact (Contact)>

<!ELEMENT SupplierParty (Party)>

<!ELEMENT ListOfOrderStatusDetailResult (OrderStatusDetailResult)>

<!ELEMENT OrderStatusDetailResult (OrderReference, OrderDate, OrderStatusDate, Status)>

<!ELEMENT OrderReference (AccountCode, BuyerRefNum, SupplierRefNum)>

<!ELEMENT AccountCode (Reference)>

<!ELEMENT Reference (RefNum)>

<!ELEMENT RefNum (#PCDATA)>

<!ELEMENT BuyerRefNum (Reference)>

<!ELEMENT SupplierRefNum (Reference)>

<!ELEMENT OrderDate (#PCDATA)>

<!ELEMENT Status (StatusNote, StatusEvent)>

<!ELEMENT StatusNote (#PCDATA*)>

<!ELEMENT StatusEvent (StatusEventCodeElement)>

<!ELEMENT StatusEventCodeElement (#PCDATA)>

<!ELEMENT OrderStatusCheckSummary (OrderStatusCheckItemErrors)>

<!ELEMENT OrderStatusCheckItemErrors (#PCDATA)>

]>
```

DOCUMENT: PriceCatalog.xml

```
<?soxtype urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0?>

<!-- A Price Catalog:  A list of product pricing information. -->

<PriceCatalog>
        <PriceCatHdr>
                <DocumentDate>19990509</DocumentDate>
                <DefaultCurrency>USD</DefaultCurrency>
                <DefaultLanguage>en</DefaultLanguage>
                <ListOfDescription>
                        <Description Lang="en">Catalog Description</Description>
                </ListOfDescription>
                <SupplierParty>
                        <Party AgencyID="CommerceOne" PartyID="1732" />
                </SupplierParty>
        </PriceCatHdr>
```

```
<ListOfPriceCatAction>
    <PriceCatAction>
        <CatalogDelete>
            <PartNum>
                <Agency AgencyID="CommerceOne" />
                <PartID>1732|1812||</PartID>
            </PartNum>
        </CatalogDelete>
    </PriceCatAction>

    <PriceCatAction>
        <PriceCatDetail>
            <PriceAction>Add</PriceAction>

            <PartNum>
                <Agency AgencyID="CommerceOne" />
                <PartID>1732|12345||</PartID>
            </PartNum>

            <ListOfDescription>
                <Description Lang="en">500 sheets white paper,
20#</Description>
            </ListOfDescription>

            <RelatedParts>
                <AdditionalIDs>
                    <ListOfPartNum>
                        <PartNum>
                            <Agency
AgencyID="AssignedByBuyer" />
                            <PartID>XYZ12345</PartID>
                        </PartNum>
                        <PartNum>
                            <Agency
AgencyID="AssignedBySupplier" />
                            <PartID>12345</PartID>
                        </PartNum>
                    </ListOfPartNum>
                </AdditionalIDs>

                <SubstituteFor>
                    <ListOfPartNum>
                        <PartNum>
                            <Agency
AgencyID="AssignedBySupplier" />
                            <PartID>12386</PartID>
                            <PartIDExt>A</PartIDExt>
                        </PartNum>
                    </ListOfPartNum>
                </SubstituteFor>

                <OtherPartNums>
                    <ListOfRelatedPartNum>
                        <RelatedPartNum
RelatedPartType="Version">
```

```
                                                    <Agency
AgencyID="AssignedBySupplier" />
                                                    <PartID>123.2</PartID>
                                                </RelatedPartNum>
                                            </ListOfRelatedPartNum>
                                        </OtherPartNums>
                                    </RelatedParts>


                                    <LeadTimeDays>5</LeadTimeDays>

                                    <LongDesc>
                                        <ListOfLangString>
                                            <LangString Lang="en">A high quality paper
product
                                                designed for professional printing.
                                            </LangString>
                                        </ListOfLangString>
                                    </LongDesc>

                                    <ListOfDescInfo>
                                        <DescInfo>
                                            <AttribCode>Color</AttribCode>
                                            <ValueCode>Red</ValueCode>
                                        </DescInfo>
                                    </ListOfDescInfo>

                                    <MinOrder>5</MinOrder>

                                    <MaxOrder>1000</MaxOrder>

                                    <LotSize>5</LotSize>



                                    <ListOfProdAttribute>
                                        <ProdAttribute>
                                            <AttribName>
                                                <LangString
Lang="en">Fabric</LangString>
                                            </AttribName>
                                            <AttribValue>
                                                <LangString Lang="en">Red</LangString>
                                            </AttribValue>
                                        </ProdAttribute>

                                        <ProdAttribute>
                                            <AttribName>
                                                <LangString Lang="en">Drawer
Height</LangString>
                                            </AttribName>
                                            <Measurement
UnitOfMeasure="FOT">1.0</Measurement>
                                        </ProdAttribute>
                                    </ListOfProdAttribute>
```

```
                                    <ListOfAttachment>
                                        <Attachment Attachment="http://www.mysite.com/xyz.gif">
                                            <Purpose>Drawing</Purpose>
                                        </Attachment>
                                    </ListOfAttachment>

                                    <ListOfKeyVal>
                                        <KeyVal Keyword="User Level">Professional</KeyVal>
                                        <KeyVal Keyword="Rating">Four Stars</KeyVal>
                                    </ListOfKeyVal>

                                    <CategoryUNSPSC>04378821</CategoryUNSPSC>

                                    <ListOfCategory>
                                        <Category>
                                            <CategoryID>Printers</CategoryID>
                                            <TreeName>SupplierTree</TreeName>
                                        </Category>
                                    </ListOfCategory>

                                    <CountryOfOrigin>US</CountryOfOrigin>

                                    <ListOfSpecialCond>
                                        <SpecialCond>
                                            <CondCode>PriceIncludesTax</CondCode>
                                        </SpecialCond>
                                    </ListOfSpecialCond>

                                    <ListOfPrice>
                                        <Price>
                                            <UnitPrice Currency="USD">1.025</UnitPrice>

        <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
                                            <QuantityRange Min="1" Max="10" />
                                        </Price>


                                        <Price>
                                            <UnitPrice Currency="USD">1.0</UnitPrice>

        <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
                                            <QuantityRange Min="11" />
                                        </Price>
                                    </ListOfPrice>
                                </PriceCatDetail>
                            </PriceCatAction>
        </ListOfPriceCatAction>
</PriceCatalog>
```

DTD: PriceCatalog.dtd

```
<!DOCTYPE PriceCatalog [
```

```
<!ELEMENT PriceCatalog (PriceCatHdr, ListOfPriceCatAction)>

<!ELEMENT PriceCatHdr (DocumentDate, DefaultCurrency, DefaultLanguage, ListOfDescription,
SupplierParty)>

<!ELEMENT DocumentDate (#PCDATA)>

<!ELEMENT DefaultCurrency (#PCDATA)>

<!ELEMENT DefaultLanguage (#PCDATA)>

<!ELEMENT ListOfDescription (Description)>

<!ELEMENT Description (#PCDATA)>
<!ATTLIST Description Lang CDATA #REQUIRED>

<!ELEMENT SupplierParty (Party)>

<!ELEMENT Party EMPTY>
<!ATTLIST Party AgencyID CDATA #REQUIRED>
<!ATTLIST Party PartyID CDATA #REQUIRED>

<!ELEMENT ListOfPriceCatAction (PriceCatAction*)>

<!ELEMENT PriceCatAction (CatalogDelete | PriceCatDetail)>

<!ELEMENT CatalogDelete (PartNum)>

<!ELEMENT PartNum (Agency, PartID, PartIDExt*)>

<!ELEMENT Agency EMPTY>
<!ATTLIST Agency AgencyID CDATA #REQUIRED>

<!ELEMENT PartID (#PCDATA)>

<!ELEMENT PriceCatDetail (PriceAction, PartNum, ListOfDescription, RelatedParts, LeadTimeDays,
LongDesc, ListOfDescInfo, MinOrder, MaxOrder, LotSize, ListOfProdAttribute, ListOfAttachment,
ListOfKeyVal, CategoryUNSPSC, ListOfCategory, CountryOfOrigin, ListOfSpecialCond, ListOfPrice)>

<!ELEMENT PriceAction (#PCDATA)>

<!ELEMENT RelatedParts (AdditionalIDs, SubstituteFor, OtherPartNums)>

<!ELEMENT AdditionalIDs (ListOfPartNum)>

<!ELEMENT ListOfPartNum (PartNum | PartNum*)>

<!ELEMENT SubstituteFor (ListOfPartNum)>

<!ELEMENT PartIDExt (#PCDATA)>

<!ELEMENT OtherPartNums (ListOfRelatedPartNum)>

<!ELEMENT ListOfRelatedPartNum (RelatedPartNum)>

<!ELEMENT RelatedPartNum (Agency, PartID)>
```

```
<!ATTLIST RelatedPartNum RelatedPartType CDATA #REQUIRED>

<!ELEMENT LeadTimeDays (#PCDATA)>

<!ELEMENT LongDesc (ListOfLangString)>

<!ELEMENT ListOfLangString (LangString)>

<!ELEMENT LangString (#PCDATA)>
<!ATTLIST LangString Lang CDATA #REQUIRED>

<!ELEMENT ListOfDescInfo (DescInfo)>

<!ELEMENT DescInfo (AttribCode, ValueCode)>

<!ELEMENT AttribCode (#PCDATA)>

<!ELEMENT ValueCode (#PCDATA)>

<!ELEMENT MinOrder (#PCDATA)>

<!ELEMENT MaxOrder (#PCDATA)>

<!ELEMENT LotSize (#PCDATA)>

<!ELEMENT ListOfProdAttribute (ProdAttribute*)>

<!ELEMENT ProdAttribute (AttribName, (AttribValue|Measurement))>

<!ELEMENT AttribName (LangString)>

<!ELEMENT AttribValue (LangString)>

<!ELEMENT Measurement (#PCDATA)>
<!ATTLIST Measurement UnitOfMeasure CDATA #REQUIRED>

<!ELEMENT ListOfAttachment (Attachment)>

<!ELEMENT Attachment (Purpose)>
<!ATTLIST Attachment Attachment CDATA #REQUIRED>

<!ELEMENT Purpose (#PCDATA)>

<!ELEMENT ListOfKeyVal (KeyVal*)>

<!ELEMENT KeyVal (#PCDATA)>
<!ATTLIST KeyVal Keyword CDATA #REQUIRED>

<!ELEMENT CategoryUNSPSC (#PCDATA)>

<!ELEMENT ListOfCategory (Category)>

<!ELEMENT Category (CategoryID, TreeName)>

<!ELEMENT CategoryID (#PCDATA)>
```

```
<!ELEMENT TreeName (#PCDATA)>

<!ELEMENT CountryOfOrigin (#PCDATA)>

<!ELEMENT ListOfSpecialCond (SpecialCond)>

<!ELEMENT SpecialCond (CondCode)>

<!ELEMENT CondCode (#PCDATA)>

<!ELEMENT ListOfPrice (Price*)>

<!ELEMENT Price (UnitPrice, UnitOfMeasure, QuantityRange)>

<!ELEMENT UnitPrice (#PCDATA)>
<!ATTLIST UnitPrice Currency CDATA #REQUIRED>

<!ELEMENT UnitOfMeasure (UOM)>

<!ELEMENT UOM (#PCDATA)>

<!ELEMENT QuantityRange EMPTY>
<!ATTLIST QuantityRange Min CDATA #REQUIRED>
<!ATTLIST QuantityRange Max CDATA #IMPLIED>

]>
```

DOCUMENT: PriceCheckRequest.xml

```
<?soxtype urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0?>

<!-- Instance of PriceCheckRequest -->

<PriceCheckRequest>
  <PriceCheckHeader>
        <!-- The supplier of the PartKeys to be quoted-->
        <SupplierID>
              <Reference>
                    <RefNum>OD1233</RefNum>
              </Reference>
        </SupplierID>

        <!-- The buyer account code -->
        <AccountCode>
              <Reference>
                    <RefNum>OD11222S</RefNum>
              </Reference>
        </AccountCode>

        <!-- The requested Currency -->
        <Currency>USD</Currency>

        <!-- The quote date -->
        <QuoteDate>19990809T01:01:01</QuoteDate>
  </PriceCheckHeader>
```

```
        <!-- A list of order items:  PartKey, quantity-->
        <ListOfBaseItemDetail>
  <BaseItemDetail>
    <LineItemNum>1</LineItemNum>
    <SupplierPartNum>
                        <PartNum>
                                <Agency AgencyID="AssignedBySupplier" />
                                <PartID>PK122122</PartID>
                        </PartNum>
    </SupplierPartNum>
    <Quantity>
      <Qty>10</Qty>
      <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
    </Quantity>
  </BaseItemDetail>

  <BaseItemDetail>
    <LineItemNum>2</LineItemNum>
    <SupplierPartNum>
                        <PartNum>
                                <Agency AgencyID="AssignedBySupplier" />
                                <PartID>PK122122</PartID>
                        </PartNum>
    </SupplierPartNum>
    <Quantity>
      <Qty>1</Qty>
      <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
    </Quantity>
  </BaseItemDetail>
        </ListOfBaseItemDetail>

</PriceCheckRequest>
```

DTD: PriceCheckRequest.dtd

```
<!DOCTYPE PriceCheckRequest [

<!ELEMENT PriceCheckRequest (PriceCheckHeader, ListOfBaseItemDetail)>

<!ELEMENT PriceCheckHeader (SupplierID, AccountCode, Currency, QuoteDate)>

<!ELEMENT SupplierID (Reference)>

<!ELEMENT Reference (RefNum)>

<!ELEMENT RefNum (#PCDATA)>

<!ELEMENT AccountCode (Reference)>

<!ELEMENT Currency (#PCDATA)>

<!ELEMENT QuoteDate (#PCDATA)>

<!ELEMENT ListOfBaseItemDetail (BaseItemDetail*)>
```

```
<!ELEMENT BaseItemDetail (LineItemNum, SupplierPartNum, Quantity)>

<!ELEMENT LineItemNum (#PCDATA)>

<!ELEMENT SupplierPartNum (PartNum)>

<!ELEMENT PartNum (Agency, PartID)>

<!ELEMENT Agency EMPTY>
<!ATTLIST Agency AgencyID CDATA #REQUIRED>

<!ELEMENT PartID (#PCDATA)>

<!ELEMENT Quantity (Qty, UnitOfMeasure)>

<!ELEMENT Qty (#PCDATA)>

<!ELEMENT UnitOfMeasure (UOM)>

<!ELEMENT UOM (#PCDATA)>

]>
```

DOCUMENT: PriceCheckResult.xml

```
<?soxtype urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0?>

<!-- Instance of PriceCheckResult -->

<PriceCheckResult>
  <PriceCheckHeader>
        <!-- The supplier of the PartKeys to be quoted-->
        <SupplierID>
              <Reference>
                    <RefNum>OD1233</RefNum>
              </Reference>
        </SupplierID>

        <!-- The buyer account code -->
        <AccountCode>
              <Reference>
                    <RefNum>OD11222S</RefNum>
              </Reference>
        </AccountCode>

        <!-- The requested Currency -->
        <Currency>USD</Currency>

        <!-- The quote date -->
        <QuoteDate>19990805T01:01:01</QuoteDate>
  </PriceCheckHeader>

        <ListOfPriceResultItem>
```

```
        <PriceResultItem>

                <!-- A list of order items:  PartKey, quantity-->
                <QuotedItem>
                        <BaseItemDetail>
                                <LineItemNum>1</LineItemNum>
                                <SupplierPartNum>
                                        <PartNum>
                                                <Agency AgencyID="AssignedBySupplier" />
                                                <PartID>PK122122</PartID>
                                        </PartNum>
                                </SupplierPartNum>
                                <Quantity>
                                        <Qty>10</Qty>
                                        <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
                                </Quantity>
                        </BaseItemDetail>
                </QuotedItem>

                <ResultPrice>
                        <Price>
                                <UnitPrice Currency="USD">19.25</UnitPrice>
                        </Price>
                </ResultPrice>

        </PriceResultItem>

        <PriceResultItem>

                <QuotedItem>
                        <BaseItemDetail>
                                <LineItemNum>2</LineItemNum>
                                <SupplierPartNum>
                                        <PartNum>
                                                <Agency AgencyID="CommerceOne" />
                                                <PartID>PK122122</PartID>
                                        </PartNum>
                                </SupplierPartNum>
                                <Quantity>
                                        <Qty>1</Qty>
                                        <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
                                </Quantity>
                        </BaseItemDetail>
                </QuotedItem>

                <ResultPrice>
                        <Price>
                                <UnitPrice Currency="USD">19.95</UnitPrice>
                        </Price>
                </ResultPrice>

        </PriceResultItem>

        </ListOfPriceResultItem>

<PriceCheckSummary>
```

```
        <PriceCheckItemErrors>0</PriceCheckItemErrors>
    </PriceCheckSummary>

</PriceCheckResult>
```

DTD: PriceCheckResult.dtd

```
<!DOCTYPE PriceCheckResult [

<!ELEMENT PriceCheckResult (PriceCheckHeader, ListOfPriceResultItem, PriceCheckSummary)>

<!ELEMENT PriceCheckHeader (SupplierID, AccountCode, Currency, QuoteDate)>

<!ELEMENT SupplierID (Reference)>

<!ELEMENT Reference (RefNum)>

<!ELEMENT RefNum (#PCDATA)>

<!ELEMENT AccountCode (Reference)>

<!ELEMENT Currency (#PCDATA)>

<!ELEMENT QuoteDate (#PCDATA)>

<!ELEMENT ListOfPriceResultItem (PriceResultItem*)>

<!ELEMENT PriceResultItem (QuotedItem, ResultPrice)>

<!ELEMENT QuotedItem (BaseItemDetail)>

<!ELEMENT BaseItemDetail (LineItemNum, SupplierPartNum, Quantity)>

<!ELEMENT LineItemNum (#PCDATA)>

<!ELEMENT SupplierPartNum (PartNum)>

<!ELEMENT PartNum (Agency, PartID)>

<!ELEMENT Agency EMPTY>
<!ATTLIST Agency AgencyID CDATA #REQUIRED>

<!ELEMENT PartID (#PCDATA)>

<!ELEMENT Quantity (Qty, UnitOfMeasure)>

<!ELEMENT Qty (#PCDATA)>

<!ELEMENT UnitOfMeasure (UOM)>

<!ELEMENT UOM (#PCDATA)>

<!ELEMENT ResultPrice (Price)>

<!ELEMENT Price (UnitPrice)>
```

```
<!ELEMENT UnitPrice (#PCDATA)>
<!ATTLIST UnitPrice Currency CDATA #REQUIRED>

<!ELEMENT PriceCheckSummary (PriceCheckItemErrors)>

<!ELEMENT PriceCheckItemErrors (#PCDATA)>

]>
```

DOCUMENT: ProductCatalog.xml

```
<?soxtype urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0?>
<!-- A Product Catalog:  A list of product information. -->
<ProductCatalog>
        <ProdCatHdr>
                <DocumentDate>19990509</DocumentDate>
                <DefaultCurrency>USD</DefaultCurrency>
                <DefaultLanguage>en</DefaultLanguage>
                <ListOfDescription>
                        <Description Lang="en">Catalog Description</Description>
                </ListOfDescription>
                <SupplierParty>
                        <Party AgencyID="CommerceOne" PartyID="1732" />
                </SupplierParty>
        </ProdCatHdr>

        <ListOfProdCatAction>
                <ProdCatAction>
                        <CatalogDelete>
                                <PartNum>
                                        <Agency AgencyID="CommerceOne" />
                                        <PartID>1732|1812||</PartID>
                                </PartNum>
                        </CatalogDelete>
                </ProdCatAction>

                <ProdCatAction>
                        <ProdCatDetail>
                                <ProdAction>Add</ProdAction>

                                <PartNum>
                                        <Agency AgencyID="CommerceOne" />
                                        <PartID>1732|12345||</PartID>
                                </PartNum>

                                <ListOfDescription>
                                        <Description Lang="en">500 sheets white paper,
20#</Description>
                                </ListOfDescription>

                                <RelatedParts>
                                        <AdditionalIDs>
                                                <ListOfPartNum>
                                                        <PartNum>
```

```
                                        <Agency
AgencyID="AssignedByBuyer" />
                                        <PartID>XYZ12345</PartID>
                                    </PartNum>
                                    <PartNum>
                                        <Agency
AgencyID="AssignedBySupplier" />
                                        <PartID>12345</PartID>
                                    </PartNum>
                                </ListOfPartNum>
                            </AdditionalIDs>

                            <SubstituteFor>
                                <ListOfPartNum>
                                    <PartNum>
                                        <Agency
AgencyID="AssignedBySupplier" />
                                        <PartID>12386</PartID>
                                        <PartIDExt>A</PartIDExt>
                                    </PartNum>
                                </ListOfPartNum>
                            </SubstituteFor>

                            <OtherPartNums>
                                <ListOfRelatedPartNum>
                                    <RelatedPartNum
RelatedPartType="Version">
                                        <Agency
AgencyID="AssignedBySupplier" />
                                        <PartID>123.2</PartID>
                                    </RelatedPartNum>
                                </ListOfRelatedPartNum>
                            </OtherPartNums>
                        </RelatedParts>


                        <LeadTimeDays>5</LeadTimeDays>

                        <LongDesc>
                            <ListOfLangString>
                                <LangString Lang="en">A high quality paper
product
                                    designed for professional printing.
                                </LangString>
                            </ListOfLangString>
                        </LongDesc>

                        <ListOfDescInfo>
                            <DescInfo>
                                <AttribCode>Color</AttribCode>
                                <ValueCode>Red</ValueCode>
                            </DescInfo>
                        </ListOfDescInfo>
```

```
                    <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>


                    <ListOfProdAttribute>
                        <ProdAttribute>
                            <AttribName>
                                <LangString
Lang="en">Fabric</LangString>
                            </AttribName>
                            <AttribValue>
                                <LangString Lang="en">Red</LangString>
                            </AttribValue>
                        </ProdAttribute>

                        <ProdAttribute>
                            <AttribName>
                                <LangString Lang="en">Drawer
Height</LangString>
                            </AttribName>
                            <Measurement
UnitOfMeasure="FOT">1.0</Measurement>
                        </ProdAttribute>
                    </ListOfProdAttribute>

                    <ListOfAttachment>
                        <Attachment Attachment="http://www.mysite.com/xyz.gif">
                            <Purpose>Drawing</Purpose>
                        </Attachment>
                    </ListOfAttachment>

                    <ListOfKeyVal>
                        <KeyVal Keyword="User Level">Professional</KeyVal>
                        <KeyVal Keyword="Rating">Four Stars</KeyVal>
                    </ListOfKeyVal>

                    <CategoryUNSPSC>04378821</CategoryUNSPSC>

                    <ListOfCategory>
                        <Category>
                            <CategoryID>Printers</CategoryID>
                            <TreeName>SupplierTree</TreeName>
                        </Category>
                    </ListOfCategory>

                    <CountryOfOrigin>US</CountryOfOrigin>

                    <ListOfSpecialCond>
                        <SpecialCond>
                            <CondCode>PriceIncludesTax</CondCode>
                        </SpecialCond>
                    </ListOfSpecialCond>

                </ProdCatDetail>
            </ProdCatAction>
        </ListOfProdCatAction>
</ProductCatalog>
```

DTD: ProductCatalog.dtd

```
<!DOCTYPE ProductCatalog [

<!ELEMENT ProductCatalog (ProdCatHdr, ListOfProdCatAction)>

<!ELEMENT ProdCatHdr (DocumentDate, DefaultCurrency, DefaultLanguage, ListOfDescription,
SupplierParty)>

<!ELEMENT DocumentDate (#PCDATA)>

<!ELEMENT DefaultCurrency (#PCDATA)>

<!ELEMENT DefaultLanguage (#PCDATA)>

<!ELEMENT ListOfDescription (Description)>

<!ELEMENT Description (#PCDATA)>
<!ATTLIST Description Lang CDATA #REQUIRED>

<!ELEMENT SupplierParty (Party)>

<!ELEMENT Party EMPTY>
<!ATTLIST Party AgencyID CDATA #REQUIRED>
<!ATTLIST Party PartyID CDATA #REQUIRED>

<!ELEMENT ListOfProdCatAction (ProdCatAction*)>

<!ELEMENT ProdCatAction (CatalogDelete | ProdCatDetail)>

<!ELEMENT CatalogDelete (PartNum)>

<!ELEMENT PartNum (Agency, PartID, PartIDExt*)>

<!ELEMENT Agency EMPTY>
<!ATTLIST Agency AgencyID CDATA #REQUIRED>

<!ELEMENT PartID (#PCDATA)>

<!ELEMENT ProdCatDetail (ProdAction, PartNum, ListOfDescription, RelatedParts, LeadTimeDays,
LongDesc, ListOfDescInfo, UnitOfMeasure, ListOfProdAttribute, ListOfAttachment, ListOfKeyVal,
CategoryUNSPSC, ListOfCategory, CountryOfOrigin, ListOfSpecialCond)>

<!ELEMENT ProdAction (#PCDATA)>

<!ELEMENT RelatedParts (AdditionalIDs, SubstituteFor, OtherPartNums)>

<!ELEMENT AdditionalIDs (ListOfPartNum)>

<!ELEMENT ListOfPartNum (PartNum | PartNum*)>

<!ELEMENT SubstituteFor (ListOfPartNum)>

<!ELEMENT PartIDExt (#PCDATA)>
```

```
<!ELEMENT OtherPartNums (ListOfRelatedPartNum)>

<!ELEMENT ListOfRelatedPartNum (RelatedPartNum)>

<!ELEMENT RelatedPartNum (Agency, PartID)>
<!ATTLIST RelatedPartNum RelatedPartType CDATA #REQUIRED>

<!ELEMENT LeadTimeDays (#PCDATA)>

<!ELEMENT LongDesc (ListOfLangString)>

<!ELEMENT ListOfLangString (LangString)>

<!ELEMENT LangString (#PCDATA)>
<!ATTLIST LangString Lang CDATA #REQUIRED>

<!ELEMENT ListOfDescInfo (DescInfo)>

<!ELEMENT DescInfo (AttribCode, ValueCode)>

<!ELEMENT AttribCode (#PCDATA)>

<!ELEMENT ValueCode (#PCDATA)>

<!ELEMENT UnitOfMeasure (UOM)>

<!ELEMENT UOM (#PCDATA)>

<!ELEMENT ListOfProdAttribute (ProdAttribute*)>

<!ELEMENT ProdAttribute (AttribName, (AttribValue|Measurement))>

<!ELEMENT AttribName (LangString)>

<!ELEMENT AttribValue (LangString)>

<!ELEMENT Measurement (#PCDATA)>
<!ATTLIST Measurement UnitOfMeasure CDATA #REQUIRED>

<!ELEMENT ListOfAttachment (Attachment)>

<!ELEMENT Attachment (Purpose)>
<!ATTLIST Attachment Attachment CDATA #REQUIRED>

<!ELEMENT Purpose (#PCDATA)>

<!ELEMENT ListOfKeyVal (KeyVal*)>

<!ELEMENT KeyVal (#PCDATA)>
<!ATTLIST KeyVal Keyword CDATA #REQUIRED>

<!ELEMENT CategoryUNSPSC (#PCDATA)>

<!ELEMENT ListOfCategory (Category)>
```

```
<!ELEMENT Category (CategoryID, TreeName)>

<!ELEMENT CategoryID (#PCDATA)>

<!ELEMENT TreeName (#PCDATA)>

<!ELEMENT CountryOfOrigin (#PCDATA)>

<!ELEMENT ListOfSpecialCond (SpecialCond)>

<!ELEMENT SpecialCond (CondCode)>

<!ELEMENT CondCode (#PCDATA)>

]>
```

DOCUMENT: PurchaseOrder.xml

```
<?soxtype urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0?>

<!--
 * Copyright (c) 1999 Commerce One. All rights reserved.  Redistribution and
 * use in source and binary forms, with or without modification, is strictly
 * prohibited without written permission from Commerce One.
 -->

<PurchaseOrder>

<OrderHeader>
   <!-- 19990805 -->
   <POIssuedDate>19990805T01:01:01</POIssuedDate>
   <RequestedDeliveryDate>19990807T01:01:01</RequestedDeliveryDate>
   <ShipByDate>19990809T01:01:01</ShipByDate>
   <OrderReference>
    <!-- An account is an agreement between a buyer and a supplier, specified
          by the account code. Remember that an agreement can consists of
          multiple contracts. Agreement is not the same as contract.
          An agreement 'contains' contract(s). -->
    <AccountCode><Reference><RefNum>CTOP</RefNum></Reference> </AccountCode>

    <!--BuyerRefNum = Buyer's PO number. -->
    <BuyerRefNum><Reference><RefNum>100</RefNum></Reference></BuyerRefNum>
    <!-- Notice I don't put the SupplierRefNum because it is optional -->
    <SupplierRefNum><Reference><RefNum>500</RefNum></Reference>
    </SupplierRefNum>
   </OrderReference>
   <OrderParty>
     <BuyerParty>
          <Party>
          <NameAddress>
            <Name1>Mr. Muljadi Sulistio</Name1>
            <Name2>Attention: Business Service Division</Name2>
            <Address1>1600 Riviera Ave</Address1>
            <Address2>Suite# 200</Address2>
            <City>Walnut Creek</City>
```

```
            <StateOrProvince>CA</StateOrProvince>
            <PostalCode>94596</PostalCode>
            <Country>US</Country>
          </NameAddress>
          <OrderContact>
            <Contact>
            <ContactName>Mr. Mike Holloway</ContactName>
            <Telephone>(925) 941-3333</Telephone>
            <Email>mike.holloway@commerceone.com</Email>
            <Fax>(925) 941-4555</Fax>
            </Contact>
          </OrderContact>
          <ReceivingContact>
            <Contact>
            <ContactName>Mr. Debbie Dub</ContactName>
            <Telephone>(925) 941-2222</Telephone>
            <Email>debbie.dub@commerceone.com</Email>
            <Fax>(925) 941-4555</Fax>
            </Contact>
          </ReceivingContact>
          <ShippingContact>
            <Contact>
            <ContactName>Ms. John Wayne</ContactName>
            <Telephone>(925) 941-1111</Telephone>
            <Email>john.wayne@commerceone.com</Email>
            <Fax>(925) 941-4555</Fax>
            </Contact>
          </ShippingContact>
          </Party>
    </BuyerParty>
    <SupplierParty>
          <Party>
          <NameAddress>
            <Name1>Millenium Supplier Corporation</Name1>
            <Name2>Attention: Office Supply Division</Name2>
            <Address1>355 Alameda Street</Address1>
            <Address2>Suite 100</Address2>
            <City>San Jose</City>
            <StateOrProvince>CA</StateOrProvince>
            <PostalCode>94588</PostalCode>
            <Country>US</Country>
          </NameAddress>
          </Party>
    </SupplierParty>
</OrderParty>
<Tax>
        <TaxPercent>000010.0000</TaxPercent>
        <Location>Walnut Creek</Location>
        <!-- TaxId is only required if category is exempt -->
        <TaxId>12345</TaxId>
        <TaxAmount>000000000000100.000</TaxAmount>
        <TaxableAmount>000000000001000.000</TaxableAmount>
</Tax>
<OrderCurrency>USD</OrderCurrency> <!-- US Dollar -->
<OrderLanguage>en</OrderLanguage> <!-- EN=English -->
```

```
    <Payment>
      <PaymentMean>CreditCard</PaymentMean>
      <PaymentTerm>Discount</PaymentTerm>
      <DiscountPercent>000005.0000</DiscountPercent>
      <DiscountDaysDue>10</DiscountDaysDue>
      <DiscountTimeRef>InvoiceDate</DiscountTimeRef>
      <NetDaysDue>30</NetDaysDue>
      <NetTimeRef>InvoiceDate</NetTimeRef>
      <CardInfo>
        <CardNum>1234432112344321</CardNum>
        <CardAuthCode>JUBF123</CardAuthCode>
        <CardRefNum>123</CardRefNum>
        <CardExpirationDate>20000805T01:01:01</CardExpirationDate>
        <CardType>AMEX</CardType>
        <CardHolderName>Mr. Joe Smith</CardHolderName>
      </CardInfo>
    </Payment>

    <PartialShipmentAllowed>true</PartialShipmentAllowed>
    <SpecialHandlingNote>Please don't shake it.</SpecialHandlingNote>
    <GeneralNote>Rush it please.</GeneralNote>
    <PartLocation>Oakland</PartLocation>

    <Transport Direction="SupplierToBuyer">
      <Mode>Air</Mode>
      <Mean>Express</Mean>
      <Carrier>Fedex</Carrier>
      <CustShippingContractNum>CTOP123</CustShippingContractNum>
      <ShippingInstruction>Please handle with care</ShippingInstruction>
    </Transport>

    <TermOfDelivery TODFunction_a="Delivery" >
      <Code>FOB</Code>
      <FOBCity>San Francisco</FOBCity>
      <FOBLocation>SFO Delta Airline Warehouse</FOBLocation>
      <FOBInstruction>Talk to the company agent in isle 2</FOBInstruction>
      <ShippingPaymentMethod>PrepaidBySeller</ShippingPaymentMethod>
    </TermOfDelivery>

    <OrderHeaderAttachment>
    <ListOfAttachment>
          <Attachment Attachment="http://www.temporary.com/Pleasantondome.doc" >
          <Purpose>BluePrint</Purpose>
          </Attachment>
    </ListOfAttachment>
    </OrderHeaderAttachment>

</OrderHeader>

<ListOfOrderDetail>
<OrderDetail>
    <BaseItemDetail>
          <LineItemNum>1</LineItemNum>
          <SupplierPartNum>
          <PartNum>
            <Agency AgencyID="AssignedBySupplier"/>
```

```
            <PartID>12345</PartID>
          </PartNum>
          </SupplierPartNum>
          <ItemDescription>Sanford Highlighting Marker</ItemDescription>
          <Quantity>
            <Qty>000000000001.000</Qty>
            <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
      </Quantity>
      <Transport Direction="SupplierToBuyer">
        <Mode>Air</Mode>
        <Mean>Express</Mean>
        <Carrier>Fedex</Carrier>
        <CustShippingContractNum>CTOP123</CustShippingContractNum>
        <ShippingInstruction>Please handle with care</ShippingInstruction>
      </Transport>
      <OffCatalogFlag>false</OffCatalogFlag>
    </BaseItemDetail>
    <BuyerExpectedUnitPrice>
      <Price><UnitPrice>00000000010.0000</UnitPrice></Price>
    </BuyerExpectedUnitPrice>
</OrderDetail>
</ListOfOrderDetail>

<OrderSummary>
    <TotalAmount>000000000001000.000</TotalAmount>
    <TotalLineNum>1</TotalLineNum>
</OrderSummary>

</PurchaseOrder>
```

DTD: PurchaseOrder.dtd

```
<!DOCTYPE PurchaseOrder [

<!ELEMENT PurchaseOrder (OrderHeader, ListOfOrderDetail, OrderSummary)>

<!ELEMENT OrderHeader (POIssuedDate, RequestedDeliveryDate, ShipByDate, OrderReference,
OrderParty, Tax, OrderCurrency, OrderLanguage, Payment, PartialShipmentAllowed,
SpecialHandlingNote, GeneralNote, PartLocation, Transport, TermOfDelivery, OrderHeaderAttachment)>

<!ELEMENT POIssuedDate (#PCDATA)>

<!ELEMENT RequestedDeliveryDate (#PCDATA)>

<!ELEMENT ShipByDate (#PCDATA)>

<!ELEMENT OrderReference (AccountCode, BuyerRefNum, SupplierRefNum)>

<!ELEMENT AccountCode (Reference)>

<!ELEMENT Reference (RefNum)>

<!ELEMENT RefNum (#PCDATA)>

<!ELEMENT BuyerRefNum (Reference)>
```

<!ELEMENT SupplierRefNum (Reference)>

<!ELEMENT OrderParty (BuyerParty, SupplierParty)>

<!ELEMENT BuyerParty (Party)>

<!ELEMENT Party (NameAddress, OrderContact*, ReceivingContact*, ShippingContact*)>

<!ELEMENT NameAddress (Name1, Name2, Address1, Address2, City, StateOrProvince, PostalCode, Country)>

<!ELEMENT Name1 (#PCDATA)>

<!ELEMENT Name2 (#PCDATA)>

<!ELEMENT Address1 (#PCDATA)>

<!ELEMENT Address2 (#PCDATA)>

<!ELEMENT City (#PCDATA)>

<!ELEMENT StateOrProvince (#PCDATA)>

<!ELEMENT PostalCode (#PCDATA)>

<!ELEMENT Country (#PCDATA)>

<!ELEMENT OrderContact (Contact)>

<!ELEMENT Contact (ContactName, Telephone, Email, Fax)>

<!ELEMENT ContactName (#PCDATA)>

<!ELEMENT Telephone (#PCDATA)>

<!ELEMENT Email (#PCDATA)>

<!ELEMENT Fax (#PCDATA)>

<!ELEMENT ReceivingContact (Contact)>

<!ELEMENT ShippingContact (Contact)>

<!ELEMENT SupplierParty (Party)>

<!ELEMENT Tax (TaxPercent, Location, TaxId, TaxAmount, TaxableAmount)>

<!ELEMENT TaxPercent (#PCDATA)>

<!ELEMENT Location (#PCDATA)>

<!ELEMENT TaxId (#PCDATA)>

<!ELEMENT TaxAmount (#PCDATA)>

<!ELEMENT TaxableAmount (#PCDATA)>

<!ELEMENT OrderCurrency (#PCDATA)>

<!ELEMENT OrderLanguage (#PCDATA)>

<!ELEMENT Payment (PaymentMean, PaymentTerm, DiscountPercent, DiscountDaysDue, DiscountTimeRef, NetDaysDue, NetTimeRef, CardInfo)>

<!ELEMENT PaymentMean (#PCDATA)>

<!ELEMENT PaymentTerm (#PCDATA)>

<!ELEMENT DiscountPercent (#PCDATA)>

<!ELEMENT DiscountDaysDue (#PCDATA)>

<!ELEMENT DiscountTimeRef (#PCDATA)>

<!ELEMENT NetDaysDue (#PCDATA)>

<!ELEMENT NetTimeRef (#PCDATA)>

<!ELEMENT CardInfo (CardNum, CardAuthCode, CardRefNum, CardExpirationDate, CardType, CardHolderName)>

<!ELEMENT CardNum (#PCDATA)>

<!ELEMENT CardAuthCode (#PCDATA)>

<!ELEMENT CardRefNum (#PCDATA)>

<!ELEMENT CardExpirationDate (#PCDATA)>

<!ELEMENT CardType (#PCDATA)>

<!ELEMENT CardHolderName (#PCDATA)>

<!ELEMENT PartialShipmentAllowed (#PCDATA)>

<!ELEMENT SpecialHandlingNote (#PCDATA)>

<!ELEMENT GeneralNote (#PCDATA)>

<!ELEMENT PartLocation (#PCDATA)>

<!ELEMENT Transport (Mode, Mean, Carrier, CustShippingContractNum, ShippingInstruction)>
<!ATTLIST Transport Direction CDATA #REQUIRED>

<!ELEMENT Mode (#PCDATA)>

<!ELEMENT Mean (#PCDATA)>

<!ELEMENT Carrier (#PCDATA)>

<!ELEMENT CustShippingContractNum (#PCDATA)>

<!ELEMENT ShippingInstruction (#PCDATA)>

<!ELEMENT TermOfDelivery (Code, FOBCity, FOBLocation, FOBInstruction, ShippingPaymentMethod)>
<!ATTLIST TermOfDelivery TODFunction_a CDATA #REQUIRED>

<!ELEMENT Code (#PCDATA)>

<!ELEMENT FOBCity (#PCDATA)>

<!ELEMENT FOBLocation (#PCDATA)>

<!ELEMENT FOBInstruction (#PCDATA)>

<!ELEMENT ShippingPaymentMethod (#PCDATA)>

<!ELEMENT OrderHeaderAttachment (ListOfAttachment)>

<!ELEMENT ListOfAttachment (Attachment)>

<!ELEMENT Attachment (Purpose)>
<!ATTLIST Attachment Attachment CDATA #REQUIRED>

<!ELEMENT Purpose (#PCDATA)>

<!ELEMENT ListOfOrderDetail (OrderDetail)>

<!ELEMENT OrderDetail (BaseItemDetail, BuyerExpectedUnitPrice)>

<!ELEMENT BaseItemDetail (LineItemNum, SupplierPartNum, ItemDescription, Quantity, Transport, OffCatalogFlag)>

<!ELEMENT LineItemNum (#PCDATA)>

<!ELEMENT SupplierPartNum (PartNum)>

<!ELEMENT PartNum (Agency, PartID)>

<!ELEMENT Agency EMPTY>
<!ATTLIST Agency AgencyID CDATA #REQUIRED>

<!ELEMENT PartID (#PCDATA)>

<!ELEMENT ItemDescription (#PCDATA)>

<!ELEMENT Quantity (Qty, UnitOfMeasure)>

<!ELEMENT Qty (#PCDATA)>

<!ELEMENT UnitOfMeasure (UOM)>

<!ELEMENT UOM (#PCDATA)>

<!ELEMENT OffCatalogFlag (#PCDATA)>

```
<!ELEMENT BuyerExpectedUnitPrice (Price)>

<!ELEMENT Price (UnitPrice)>

<!ELEMENT UnitPrice (#PCDATA)>

<!ELEMENT OrderSummary (TotalAmount, TotalLineNum)>

<!ELEMENT TotalAmount (#PCDATA)>

<!ELEMENT TotalLineNum (#PCDATA)>

]>
```

DOCUMENT: PurchaseOrderResponse.xml

```
<?soxtype urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0?>

<!--
 * Copyright (c) 1999 Commerce One. All rights reserved.  Redistribution and
 * use in source and binary forms, with or without modification, is strictly
 * prohibited without written permission from Commerce One.
 -->

<!-- This is a sample response of the po1.xml document.
    This sample represents the situation when supplier accepts
    the PurchaseOrder as is. -->

<PurchaseOrderResponse>

<OrderResponseHeader>

   <POIssuedDate>19990805T01:01:01</POIssuedDate>
   <RequestedDeliveryDate>19990807T01:01:01</RequestedDeliveryDate>
   <ShipByDate>19990809T01:01:01</ShipByDate>
   <OrderReference>
    <!-- An account is an agreement between a buyer and a supplier, specified
          by the account code. Remember that an agreement can consists of
          multiple contracts. Agreement is not the same as contract.
          An agreement 'contains' contract(s). -->
    <AccountCode><Reference><RefNum>CTOP</RefNum></Reference> </AccountCode>

    <!--BuyerRefNum = Buyer's PO number. -->
    <BuyerRefNum><Reference><RefNum>100</RefNum></Reference></BuyerRefNum>
    <!-- Notice I don't put the SupplierRefNum because it is optional -->
    <SupplierRefNum><Reference><RefNum>500</RefNum></Reference>
    </SupplierRefNum>
   </OrderReference>
   <OrderParty>
     <BuyerParty>
          <Party>
          <NameAddress>
            <Name1>Mr. Muljadi Sulistio</Name1>
            <Name2>Attention: Business Service Division</Name2>
            <Address1>1600 Riviera Ave</Address1>
```

```
            <Address2>Suite# 200</Address2>
            <City>Walnut Creek</City>
            <StateOrProvince>CA</StateOrProvince>
            <PostalCode>94596</PostalCode>
            <Country>US</Country>
          </NameAddress>
          <OrderContact>
            <Contact>
            <ContactName>Mr. Mike Holloway</ContactName>
            <Telephone>(925) 941-3333</Telephone>
            <Email>mike.holloway@commerceone.com</Email>
            <Fax>(925) 941-4555</Fax>
            </Contact>
          </OrderContact>
          <ReceivingContact>
            <Contact>
            <ContactName>Mr. Debbie Dub</ContactName>
            <Telephone>(925) 941-2222</Telephone>
            <Email>debbie.dub@commerceone.com</Email>
            <Fax>(925) 941-4555</Fax>
            </Contact>
          </ReceivingContact>
          <ShippingContact>
            <Contact>
            <ContactName>Ms. John Wayne</ContactName>
            <Telephone>(925) 941-1111</Telephone>
            <Email>john.wayne@commerceone.com</Email>
            <Fax>(925) 941-4555</Fax>
            </Contact>
          </ShippingContact>
          </Party>
    </BuyerParty>
    <SupplierParty>
          <Party>
          <NameAddress>
            <Name1>Millenium Supplier Corporation</Name1>
            <Name2>Attention: Office Supply Division</Name2>
            <Address1>355 Alameda Street</Address1>
            <Address2>Suite 100</Address2>
            <City>San Jose</City>
            <StateOrProvince>CA</StateOrProvince>
            <PostalCode>94588</PostalCode>
            <Country>US</Country>
          </NameAddress>
          </Party>
    </SupplierParty>
</OrderParty>
<Tax>
        <TaxPercent>000010.0000</TaxPercent>
        <Location>Walnut Creek</Location>
        <!-- TaxId is only required if category is exempt -->
        <TaxId>12345</TaxId>
        <TaxAmount>000000000000100.000</TaxAmount>
        <TaxableAmount>000000000001000.000</TaxableAmount>
</Tax>
<OrderCurrency>USD</OrderCurrency> <!-- US Dollar -->
```

```
    <OrderLanguage>en</OrderLanguage> <!-- EN=English -->

    <Payment>
      <PaymentMean>CreditCard</PaymentMean>
      <PaymentTerm>Discount</PaymentTerm>
      <DiscountPercent>000005.0000</DiscountPercent>
      <DiscountDaysDue>10</DiscountDaysDue>
      <DiscountTimeRef>InvoiceDate</DiscountTimeRef>
      <NetDaysDue>30</NetDaysDue>
      <NetTimeRef>InvoiceDate</NetTimeRef>
      <CardInfo>
        <CardNum>1234432112344321</CardNum>
        <CardAuthCode>JUBF123</CardAuthCode>
        <CardRefNum>123</CardRefNum>
        <CardExpirationDate>20000805T01:01:01</CardExpirationDate>
        <CardType>AMEX</CardType>
        <CardHolderName>Mr. Joe Blow</CardHolderName>
      </CardInfo>
    </Payment>

    <PartialShipmentAllowed>true</PartialShipmentAllowed>
    <SpecialHandlingNote>Please don't shake it.</SpecialHandlingNote>
    <GeneralNote>Rush it please.</GeneralNote>
    <PartLocation>Oakland</PartLocation>

    <Transport Direction="SupplierToBuyer">
      <Mode>Air</Mode>
      <Mean>Express</Mean>
      <Carrier>Fedex</Carrier>
      <CustShippingContractNum>CTOP123</CustShippingContractNum>
      <ShippingInstruction>Please handle with care</ShippingInstruction>
    </Transport>

    <TermOfDelivery TODFunction_a="Delivery" >
      <Code>FOB</Code>
      <FOBCity>San Francisco</FOBCity>
      <FOBLocation>SFO Delta Airline Warehouse</FOBLocation>
      <FOBInstruction>Talk to the company agent in isle 2</FOBInstruction>
      <ShippingPaymentMethod>PrepaidBySeller</ShippingPaymentMethod>
    </TermOfDelivery>

    <OrderHeaderAttachment>
          <ListOfAttachment>
          <Attachment Attachment="http://www.temporary.com/Pleasantondome.doc" >
          <Purpose>BluePrint</Purpose>
          </Attachment>
          </ListOfAttachment>
    </OrderHeaderAttachment>

    <!-- In this particular PO response, we accept the PO as is. -->
    <ActionRequestOrNotification>AcceptedAsIs</ActionRequestOrNotification>

</OrderResponseHeader>

<ListOfOrderResponseDetail>
<OrderResponseDetail>
```

```
   <BaseItemDetail>
          <LineItemNum>1</LineItemNum>
          <SupplierPartNum>
          <PartNum>
             <Agency AgencyID="AssignedBySupplier"/>
             <PartID>12345</PartID>
          </PartNum>
          </SupplierPartNum>
          <ItemDescription>Sanford Highlighting Marker</ItemDescription>
          <Quantity>
            <Qty>000000000001.000</Qty>
            <UnitOfMeasure><UOM>EA</UOM></UnitOfMeasure>
      </Quantity>
      <Transport Direction="SupplierToBuyer">
         <Mode>Air</Mode>
         <Mean>Express</Mean>
         <Carrier>Fedex</Carrier>
         <CustShippingContractNum>CTOP123</CustShippingContractNum>
         <ShippingInstruction>Please handle with care</ShippingInstruction>
      </Transport>
      <OffCatalogFlag>false</OffCatalogFlag>
   </BaseItemDetail>
   <BuyerExpectedUnitPrice>
      <Price><UnitPrice>00000000010.0000</UnitPrice></Price>
   </BuyerExpectedUnitPrice>
</OrderResponseDetail>
</ListOfOrderResponseDetail>

<OrderResponseSummary>
   <TotalAmount>000000000001000.000</TotalAmount>
   <TotalLineNum>1</TotalLineNum>
</OrderResponseSummary>

</PurchaseOrderResponse>
```

DTD: PurchaseOrderResponse.dtd

```
<!DOCTYPE PurchaseOrderResponse [

<!ELEMENT PurchaseOrderResponse (OrderResponseHeader, ListOfOrderResponseDetail,
OrderResponseSummary)>

<!ELEMENT OrderResponseHeader (POIssuedDate, RequestedDeliveryDate, ShipByDate,
OrderReference, OrderParty, Tax, OrderCurrency, OrderLanguage, Payment, PartialShipmentAllowed,
SpecialHandlingNote, GeneralNote, PartLocation, Transport, TermOfDelivery, OrderHeaderAttachment,
ActionRequestOrNotification)>

<!ELEMENT POIssuedDate (#PCDATA)>

<!ELEMENT RequestedDeliveryDate (#PCDATA)>

<!ELEMENT ShipByDate (#PCDATA)>

<!ELEMENT OrderReference (AccountCode, BuyerRefNum, SupplierRefNum)>
```

<!ELEMENT AccountCode (Reference)>

<!ELEMENT Reference (RefNum)>

<!ELEMENT RefNum (#PCDATA)>

<!ELEMENT BuyerRefNum (Reference)>

<!ELEMENT SupplierRefNum (Reference)>

<!ELEMENT OrderParty (BuyerParty, SupplierParty)>

<!ELEMENT BuyerParty (Party)>

<!ELEMENT Party (NameAddress, OrderContact*, ReceivingContact*, ShippingContact*)>

<!ELEMENT NameAddress (Name1, Name2, Address1, Address2, City, StateOrProvince, PostalCode, Country)>

<!ELEMENT Name1 (#PCDATA)>

<!ELEMENT Name2 (#PCDATA)>

<!ELEMENT Address1 (#PCDATA)>

<!ELEMENT Address2 (#PCDATA)>

<!ELEMENT City (#PCDATA)>

<!ELEMENT StateOrProvince (#PCDATA)>

<!ELEMENT PostalCode (#PCDATA)>

<!ELEMENT Country (#PCDATA)>

<!ELEMENT OrderContact (Contact)>

<!ELEMENT Contact (ContactName, Telephone, Email, Fax)>

<!ELEMENT ContactName (#PCDATA)>

<!ELEMENT Telephone (#PCDATA)>

<!ELEMENT Email (#PCDATA)>

<!ELEMENT Fax (#PCDATA)>

<!ELEMENT ReceivingContact (Contact)>

<!ELEMENT ShippingContact (Contact)>

<!ELEMENT SupplierParty (Party)>

<!ELEMENT Tax (TaxPercent, Location, TaxId, TaxAmount, TaxableAmount)>

<!ELEMENT TaxPercent (#PCDATA)>

<!ELEMENT Location (#PCDATA)>

<!ELEMENT TaxId (#PCDATA)>

<!ELEMENT TaxAmount (#PCDATA)>

<!ELEMENT TaxableAmount (#PCDATA)>

<!ELEMENT OrderCurrency (#PCDATA)>

<!ELEMENT OrderLanguage (#PCDATA)>

<!ELEMENT Payment (PaymentMean, PaymentTerm, DiscountPercent, DiscountDaysDue, DiscountTimeRef, NetDaysDue, NetTimeRef, CardInfo)>

<!ELEMENT PaymentMean (#PCDATA)>

<!ELEMENT PaymentTerm (#PCDATA)>

<!ELEMENT DiscountPercent (#PCDATA)>

<!ELEMENT DiscountDaysDue (#PCDATA)>

<!ELEMENT DiscountTimeRef (#PCDATA)>

<!ELEMENT NetDaysDue (#PCDATA)>

<!ELEMENT NetTimeRef (#PCDATA)>

<!ELEMENT CardInfo (CardNum, CardAuthCode, CardRefNum, CardExpirationDate, CardType, CardHolderName)>

<!ELEMENT CardNum (#PCDATA)>

<!ELEMENT CardAuthCode (#PCDATA)>

<!ELEMENT CardRefNum (#PCDATA)>

<!ELEMENT CardExpirationDate (#PCDATA)>

<!ELEMENT CardType (#PCDATA)>

<!ELEMENT CardHolderName (#PCDATA)>

<!ELEMENT PartialShipmentAllowed (#PCDATA)>

<!ELEMENT SpecialHandlingNote (#PCDATA)>

<!ELEMENT GeneralNote (#PCDATA)>

<!ELEMENT PartLocation (#PCDATA)>

<!ELEMENT Transport (Mode, Mean, Carrier, CustShippingContractNum, ShippingInstruction)>
<!ATTLIST Transport Direction CDATA #REQUIRED>

<!ELEMENT Mode (#PCDATA)>

<!ELEMENT Mean (#PCDATA)>

<!ELEMENT Carrier (#PCDATA)>

<!ELEMENT CustShippingContractNum (#PCDATA)>

<!ELEMENT ShippingInstruction (#PCDATA)>

<!ELEMENT TermOfDelivery (Code, FOBCity, FOBLocation, FOBInstruction, ShippingPaymentMethod)>
<!ATTLIST TermOfDelivery TODFunction_a CDATA #REQUIRED>

<!ELEMENT Code (#PCDATA)>

<!ELEMENT FOBCity (#PCDATA)>

<!ELEMENT FOBLocation (#PCDATA)>

<!ELEMENT FOBInstruction (#PCDATA)>

<!ELEMENT ShippingPaymentMethod (#PCDATA)>

<!ELEMENT OrderHeaderAttachment (ListOfAttachment)>

<!ELEMENT ListOfAttachment (Attachment)>

<!ELEMENT Attachment (Purpose)>
<!ATTLIST Attachment Attachment CDATA #REQUIRED>

<!ELEMENT Purpose (#PCDATA)>

<!ELEMENT ActionRequestOrNotification (#PCDATA)>

<!ELEMENT ListOfOrderResponseDetail (OrderResponseDetail)>

<!ELEMENT OrderResponseDetail (BaseItemDetail, BuyerExpectedUnitPrice)>

<!ELEMENT BaseItemDetail (LineItemNum, SupplierPartNum, ItemDescription, Quantity, Transport, OffCatalogFlag)>

<!ELEMENT LineItemNum (#PCDATA)>

<!ELEMENT SupplierPartNum (PartNum)>

<!ELEMENT PartNum (Agency, PartID)>

<!ELEMENT Agency EMPTY>
<!ATTLIST Agency AgencyID CDATA #REQUIRED>

<!ELEMENT PartID (#PCDATA)>

<!ELEMENT ItemDescription (#PCDATA)>

<!ELEMENT Quantity (Qty, UnitOfMeasure)>

<!ELEMENT Qty (#PCDATA)>

<!ELEMENT UnitOfMeasure (UOM)>

<!ELEMENT UOM (#PCDATA)>

<!ELEMENT OffCatalogFlag (#PCDATA)>

<!ELEMENT BuyerExpectedUnitPrice (Price)>

<!ELEMENT Price (UnitPrice)>

<!ELEMENT UnitPrice (#PCDATA)>

<!ELEMENT OrderResponseSummary (TotalAmount, TotalLineNum)>

<!ELEMENT TotalAmount (#PCDATA)>

<!ELEMENT TotalLineNum (#PCDATA)>

]>

LIST OF REFERENCES

[1] A. Y. Levy, "The information manifold approach to data integration," *IEEE Intelligent Systems*, vol. 13, pp. 12-16, 1998.

[2] W. W. Cohen, "The Whirl approach to data integration," *IEEE Intelligent Systems*, vol. 13, pp. 20-24, 1998.

[3] J. Widom, "Research problems in data warehousing," presented at Fourth International Conference on Information and Knowledge Management, Baltimore, MD, 1995.

[4] J. Widom, "Integrating heterogeneous databases: lazy or eager?" *ACM Computing Surveys*, vol. 28A, pp. 52-57, 1996.

[5] G. Wiederhold, "Mediators in the Architecture of Future Information Systems," *IEEE Computer*, vol. 25, pp. 38-49, 1992.

[6] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom, "Object exchange across heterogeneous information sources," presented at Eleventh International Conference on Data Engineering, Taipei, Taiwan, 1995.

[7] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, and Y. Zhuge, "The Stanford data warehousing project," *Bulletin of the Technical Committee on Data Engineering*, vol. 18, pp. 41-48, 1995.

[8] T. Lahiri, S. Abiteboul, and J. Widom. “Ozone: Integrating structured and semistructured data,” Proceedings of the Seventh International Conference on Database Programming Languages, Kinloch Rannoch, Scotland, September 1999.

[9] J. McHugh and J. Widom, “Integrating dynamically-fetched external information into a DBMS for semistructured data”, *SIGMOD Record*, vol. 26, no. 4, pp. 24-31, December 1997. Also appeared in Proceedings of the Workshop on Management of Semistructured Data, pages 75-82, Tucson, Arizona, May 1997.

[10] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. “Lore: A database management system for semistructured data,” *SIGMOD Record*, vol. 26, no. 3, pp. 54-66, September 1997.

[11] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. "The Lorel query language for semistructured data", *International Journal on Digital Libraries*, vol. 1, no. 1, pp. 68-88, April 1997.

[12] World Wide Web Consortium, "Extensible markup language (XML) 1.0," http://www.w3.org/TR/1998/REC-xml-19980210 1998.

[13] R. Goldman, J. McHugh, and J. Widom, "From semistructured data to XML: migrating the Lore data model and query language", Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99), pp. 25-30, Philadelphia, June 1999.

[14] R. Goldman, J. McHugh, and J. Widom. "Lore: A database management system for XML," *Dr. Dobb's Journal*, vol. 25, no. 4, pp. 76-80, April 2000.

[15] J. Widom, "Data management for XML", *IEEE Data Engineering Bulletin*, Special Issue on XML, vol. 22, no. 3, pp. 44-52, September 1999.

[16] J. Shanmuygasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, J Naughton, "Relational databases for querying XML documents: Limitations and opportunities," Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

[17] A. Deutsch, M. Fernandez, D. Suciu, "Storing semistructured data with STORED," Proceedings of SIGMOD, Philadephia, 1999.

[18] M. Fernandez, D. Suciu, "Optimizing regular path expressions using graph schemas", Proceedings of ICDT, Delphi, Greece, 1997.

[19] R. Goldman and J. Widom. "DataGuides: Enabling query formulation and optimization in semistructured databases", Proceedings of the Twenty-Third International Conference on Very Large Data Bases, pp. 436-445, Athens, Greece, August 1997.

[20] R. Goldman and J. Widom. "Approximate DataGuides," Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, Jerusalem, Israel, January 1999.

[21] R. S. de Oliveira, "Resolving structural conflicts during integration of XML data sources," University of Florida, Gainesville, Archives (Non-Circulating) LD1780 1999 .O473.

[22] M. Sipser, Introduction to the Theory of Computation, PWS Publishing Company, Boston, 1997.

[23] R. Goldman, S. Chawathe, A. Crespo, J. McHugh, “A standard textual interchange format for the Object Exchange Model (OEM),” Technical Report, Stanford University, October, 1996.

[24] Y. Papakonstaninou, H. Garcia-Molina and J. Widom, “Object exchange across heterogeneous information sources,” Proceedings of the Eleventh International Conference on Data Engineering, pp. 251-260, Taipei, Taiwan, Mardch 1995.

[25] B. DuCharme, XML: The Annotated Specification, Prentice Hall PTR, Upper Saddle River, NJ, 1999.

[26] E. R. Harold, XML: Extensible Markup Language, IDG Books Worldwide, Foster City, CA, 1998.

[27] E. R. Harold, The XML Bible, IDG Books Worldwide, Foster City, CA, 1999.

[28] D. Esposito, Cutting Edge: XML Languages, Microsoft Internet Developer, Redmond, WA, June, 1999.

[29] Microsoft Corporation, “XML: A technical perspective,” http://msdn.microsoft.com/xml/articles/xmlwhite.asp. 1998.

[30] M. Edwards, “XML: Data the Way You Want It,” http://msdn.microsoft.com/xml/articles/xmldata.asp, 1997.

[31] C. Heinemann, “Going from HTML to XML,” http://msdn.microsoft.com/xml/articles/xmldata.asp, 1998.

[32] World Wide Web Consortium, “XML-QL: A Query Language for XML,” http://www.w3.org/TR/1998/NOTE-xml-ql-19980819, 1998.

[33] World Wide Web Consortium, “Document Object Model (DOM) Level 1 Specification,” http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, 1998.

[34] D. Megginson, “SAX 1.0: The Simple API for XML,” http://www.megginson.com/SAX/ Members of the XML-DEV mailing list, ed., 1998.

[35] E. Myers, “An O(ND) difference algorithm and its variations”, *Algorithmica*, vol. 1, no. 2, pp. 251-266, 1986.

[36] M. Kifer, “EDIFF--A Comprehensive Interface to diff for Emacs 19,” ftp://ftp.cs.synysb.edu, 1995.

[37] A. van Hoff, J. Payne, "Generic Diff Format Specification," a submission to W3C from Marimba, 1997, http://www.w3.org/TR/NOTE-gdiff-19970901.

[38] W. Labio and H. Garcia-Molina, "Efficient Algorithms to Compare Snapshots," ftp://db.standord.edu/pub/labio/1995/, 1995.

[39] D. Shasha and K. Zhang, "Fast algorithms for the unit cost editing distance between trees", *Journal of Algorithms*, vol. 11, pp. 581-621, 1990.

[40] S. Williams, "HTTP: Delta-Encoding Notes," http://ei.cs.vt.edu/~williams/DIFF/prelim.html, 1997.

[41] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, "Potential benefits of delta encoding and data compression for HTTP," Proceedings SIGCOMM '97, Cannes, France, 1997.

[42] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, "Change detection in hierarchically structured information", Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 493-504, Montreal, Canada, June 1996.

[43] B. Ludaescher, Y. Papakonstantinou, P. Velikhov, V. Vianu "View definition and DTD inference for XML," Post-ICDT Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, Jerusalem, 1999.

[44] S. Nestorov, S. Abiteboul and R. Motwani, "Extracting schema from semi-structured data", ACM SIGMOD International Conference on Management of Data, 1998.

[45] D. Angluin, "On the complexity of minimum inference of regular sets," *Information and Control*, vol. 39, no. 3, pp. 337-350, December 1978.

[46] S. Ginsburg, The Mathematical Theory of Context-Free Languages, McGraw-Hill, New York, 1966.

[47] J. E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.

BIOGRAPHICAL SKETCH

Hongyu Guo received his bachelor's and master's degrees in theoretical physics in Nankai University in China. He was a lecturer in Hebei Institute of Technology before he came to the United States. He has done extensive research on elementary particle theory and statistical physics. He also has a deep interest in computer science. He studied in the Database R&D Center at CISE department in the University of Florida during 1997-2000 and will be receiving his MS degree in computer science in August 2000. His research interest at the University of Florida has been focused on database integration systems, XML DTD inference and their e-commerce applications.