<<DIR>>-XML^2 - Unambiguous Access to XML-based Business Documents in B2B E-Commerce

Christian Huemer
University of Vienna
Liebiggasse 4/3-4
1010 Vienna, Austria
+43 1 4277 38443
christian.huemer@univie.ac.at

ABSTRACT

XML-based vocabularies have become more and more popular in application-to-application exchanges. Like traditional EDI standards, XML-based formats require an additional agreement on rules to access XML-based data structures. It is our goal to define a machine-readable format for these agreements to accompany XML DTDs or schemas. We present our approach by using a simplified state machine to define access rules. The semantics of the state machine will be expressed by the means of XML itself to be stored in a registry.

Categories and Subject Descriptors

H.5.3 [**INFORMATION INTERFACES AND PRESENTATION**]: Group and Organization Interfaces – *evaluation/methodology, theory and models, web-based interaction.*

General Terms

Standardization, Languages.

Keywords

B2B e-commerce, XML, EDI, UN/EDIFACT, xCBL, ebXML.

1. INTRODUCTION

B2B systems already existed before the Internet hype. Since the 1960ies, organizations have been exchanging business information with their business partners by means of electronic data interchange (EDI). EDI can be defined as the application-to-application exchange of standard business documents between companies, independent of software, hardware, and communication networks [1]. Over the past decades various EDI standards capturing syntax and semantics of business documents have been developed: Proprietary formats, sector specific solutions, and branch-independent standards like X12 and UN/EDIFACT. All these formats more or less have two things in common: First, they use an implicit data identification mechanism. Second, they are successfully used by large organizations, but fail in the acceptance of small and medium enterprises (SMEs).

On the contrary, XML uses an explicit data identification mechanism by tagging the information and is said to be accepted even by SMEs. However, today's XML-based business vocabularies inherit a weak point of traditional EDI standards which is further detailed in Section 2: the generic definition of the document types. This means that the document types are not capable of capturing all constraints and dependencies among the included components. Accordingly, producing a well-formed and valid XML document does not guarantee that the receiver supporting the document type is able to process it. In traditional EDI systems additional rules on the document structure - called Message Implementation Guidelines (MIGs) - are agreed on by the business partners and are implemented on each partner's side prior to the interchanges. The implementation of these off-line agreements make EDI difficult and costly to set-up and maintain.

It is our goal to define a simple set of rules, similar to those used in MIGs, to define the access to XML business documents. Furthermore, it should be guaranteed that these rules can be expressed in a machine-readable format, ideally in XML itself. Since we do not stick to a particular business vocabulary used for the exchange, it is a precondition that the organizations support the document type and understand the semantics of its components. Consequently, we do not specify any rules on how to interpret and process the document. In Section 3 we present our approach based on UML state machines that allow a graphical representation understood by business experts and a representation in a machine-readable format.

2. PROBLEMS OF B2B VOCABULARIES

In this Section we analyze the problems resulting from a generic structure approach and validate them by the example of specifying parties in a purchase order. We emphasize on the fact that the major B2B problems are independent of a transfer-syntax. For demonstration purposes we have selected UN/EDIFACT as representative of traditional EDI standards and xCBL as representative of XML-based vocabularies.

UN/EDIFACT is the mostly used traditional standard today. In order to be globally valid, message types represent an assembly of all the data fields of any business in the world. However, only a very few data elements of the message type are used in a particular

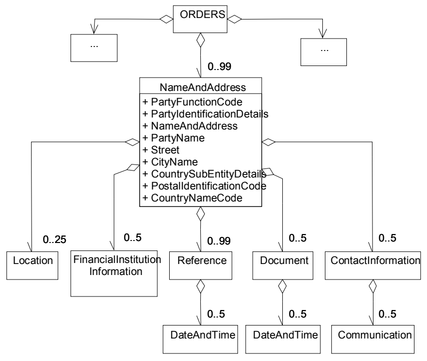


Figure 1. Parties in a UN/EDIFACT purchase order

business transaction. Thus, EDI branch organizations specify MIGs by trimming down the overloaded message types to suit the requirements of business organizations in a particular sector and/or particular part of the world [6]. Before organizations, moreover, can start doing business on-line they have to agree on an even more specific MIG which considers the requirements and limitations of their information systems.

For the purpose of a syntax-neutral analysis, Figure 1 depicts a UML class diagram representing the structure of parties in a UN/EDIFACT standard message type. Accordingly, there is no need to specify any party in a purchase order, but one can specify up to 99 parties as instances of *NameAndAddress*. The party type is identified by a code in the *PartyFunctionCode* attribute. Theoretically, each party type a code exists for might be transmitted. However, not all of them are relevant for a purchase order, e.g. the *Social Security Collectors Office*. Some are useful in a purchase order like a *Buyer's Agent*, but might not be processable for a particular organization. Accordingly, a MIG has to specify all the different party types that are meaningful.

There are multiple ways of identifying a party in an interchange. One can use a unique party identification in the *Party-IdentificationDetails*, unstructured name and address lines in the *NameAndAddress* attribute, or use the remaining attributes *Party Name*, *Street*, *City Name*, etc. for a more structured description. The MIG has to specify which way to use. Furthermore, the way it has to be done might be dependent on the party type.

Additionally, locations, financial institutions, references, documents, and contacts can be specified for each party. The MIG has to identify which of them are meaningful. But again it is not enough to declare contacts as meaningful. It is necessary to identify which kind of contact is meaningful, which attributes to use for which kind of contact, and again how this all might differ for each type of party.

It follows that producing a UN/EDIFACT message conforming to the standard does not mean that the receiver of this message supporting UN/EDIFACT is able to process it. Consequently, a message must be in conformance to a MIG which defines additional access rules to ensure interoperability.

The question is now whether XML vocabularies by itself specify unambiguous interchange definitions or whether there is also a need for MIGs. We choose the (extensible) Common Business Library 2.0 or xCBL [2], one of the most popular XML-based approaches, for illustration. Figure 2 uses UML to depict the structure of parties in an xCBL purchase order document type.

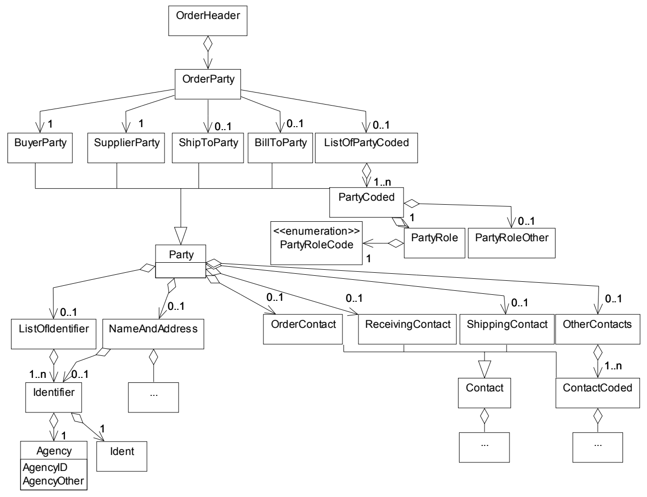


Figure 2. Parties in an xCBL purchase order

xCBL explicitly mentions the buyer, the seller, the party, the goods to ship to, and the party to be billed as meaningful parties in a purchase order. According to the definition, the buyer and the seller information is mandatory even if there is no business need to specify e.g. the seller. The *ShipToParty* and the *BillToParty* are optional. Further parties could be used according to a list of 'coded' parties. There already exists an enumeration of different useful types (like *StoreNumber* or *SupplierAgent*), but without any rules concerning their usage.

In xCBL, there are also multiple ways of identifying a party. It is possible to use one or more identifiers for a party. If only one identifier is used it could be specified in the *PartyID* attribute of *Party* or in the *Identifier* element in (a one element) *ListOfIdentifiers*. If more than one identifier is indicated, the concept of *ListOfIdentifiers* must be used. Alternatively, a party can be specified by using the structures under *NameAndAddress*.

Contact persons can be included for all types of parties. But again there is no rule to define which types of contact are useful for which type of party, not to mention the use of different attributes for each type of contact. xCBL does not support constraints to be specified according to roles of different components.

To sum up, xCBL has been developed and modeled after EDI semantics to preserve and extend the EDI investments. It has also adopted the ambiguous definitions of generic document structures. These document structures, and not the syntax as quite commonly stated, cause the major problems in B2B e-commerce. Like xCBL, any serious e-commerce language will have similar problems, because they will take advantage of reusing components. Most of the described problems are a result of reusing different components in a slightly different way.

3. ACCESSING BUSINESS DOCUMENTS

It is our goal to find an approach in which no additional off-line agreements between partners on a certain view on a document structure is needed. Our approach follows the Open-edi reference model [5] by distinguishing a business operational view (BOV) and a functional service view (FSV). BOV standards define the business independent of any transfer-syntax, whereas the FSV standards map these definitions into a certain transfer-syntax. This separation is also envisioned by the ebXML initiative [3], which our approach could fit into. In ebXML, business documents supporting a business activity are built by core components and are modeled according to UN/CEFACT's modeling methodology (UMM) [4] in a UML class diagram using common business objects (CBOs). In our approach we concentrate on the BOV by using state machines to depict access rules to the CBOs of business documents. The graphical presentation should enable a verification by business experts.

In absence of an existing set of ebXML core components (which will further be developed in a UN/CEFACT project), we make assumptions about CBOs to be used. Note that these assumptions do not influence the overall approach, since the exact definition of individual CBOs is not of importance. It is only important that agreed CBOs exist, because we do not care about semantic definitions of CBOs and their attributes. It is a precondition that the business partners understand the semantics of the CBOs.

In Figure 3 we assume the CBOs *Date*, *Party*, *Address* and *Contact* among others to build a purchase order. This document structure would be capable of capturing all the business data as stated in the following extract of the business requirements: The purchase order must have a unique purchase order identification and must include the date the purchase order is placed. The buyer must identify itself either by the buyer identification assigned by the seller or by its full address. This address is also used for billing. The buyer can indicate multiple delivery addresses, P.O. boxes are not useful in a delivery address. Furthermore, the buyer might specify multiple contact persons to be consulted for questions regarding the purchase order. The seller information - represented by a buyer assigned identification - is optional.

Each CBO definition must include a defined set of roles the CBO can take on. We use these roles on the aggregation relationships between the CBOs to define the document structure. This allows the unambiguous specification of aggregations between CBOs in different roles. A solution using roles on aggregation hierarchies is presented in Figure 3.

To define how each role accesses its (UML) attributes and its related CBOs, a UML state machine is used. The state machine has to define more or less the semantics found in a MIG:

- what happens if a value for an attribute is specified or not
- what happens if an attribute is of a certain value or not
- what happens if access to another CBO succeeds or fails

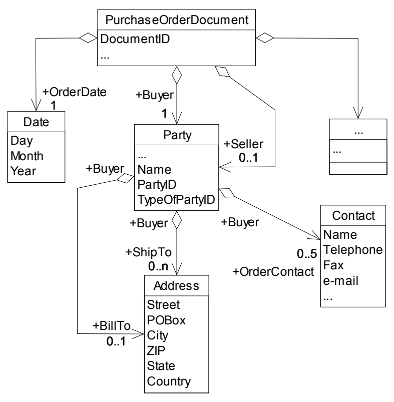


Figure 3. Class diagram using roles on CBOs

Hence, each state in the state machine is either a pseudo state (start and end state) or represents the access to an attribute, a related CBO or an internal parameter. The transitions reflect the result of the access. The guard of the transition is used to control the transition according to the result. Therefore, conditions to be verified by guards are: (1) was the access successful or did it fail, (2) is the result of a certain value or not, and (3) is an internal parameter of a certain value. Furthermore, an action can be assigned to a transition. This action is used to set values of internal parameters. These internal parameters can be accessed either within a state or can be part of a condition specified in a guard. If a state accesses another CBO in a defined role, the state uses a submachine to show the access within the other type/role of CBO. Each state machine has end states of value *OK* or *FAIL*. The transitions going out from the access state in the supermachine can verify whether the access was successful or failed. The value corresponds to the end state of the submachine. This concept guarantees traceability of the access rules of the overall document. In Figure 4 we present an extract of the state machines for our purchase order document example. The left side shows the state machine for the overall document, whereas the right side depicts the submachine to access the buyer information.

The semantics being captured by the state machine are shown in a class diagram on the left side of Figure 5. The class diagram is based on the UML metamodel of state machines [7]. We have eliminated those classes of the UML metamodel handling concepts not being relevant for our approach. Furthermore, we have added some classes to reflect the state machine's purpose to access CBOs. A central class is *CBOUsage*, which expresses the use of a CBO in the context of a certain type/role. Each *CBOUsage* uses other CBOs in a certain context or, in other words, a *CBOUsage* can be composed of other *CBOUsage*s. How a *CBOUsage* actually uses a CBO is shown by a state machine. Therefore there is a one-to-one relationship between the *CBOUsage* and the *AccessStateMachine*. Each state machine comprises multiple states. Each state might be reached by multiple transitions and lead to multiple outgoing transitions. There can be a guard specifying a condition for each transition. In our case only a transition can lead to a certain action. Each state machine can control multiple internal parameters. We distinguish between three types of states: pseudo states are start and end states. Simple states either access an attribute of a CBO or an internal parameter of the state machine. Submachine states access another role/type of CBO, which is shown by another state machine. There is a

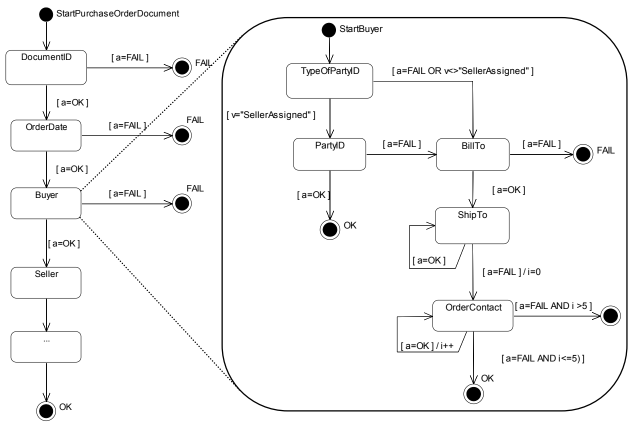
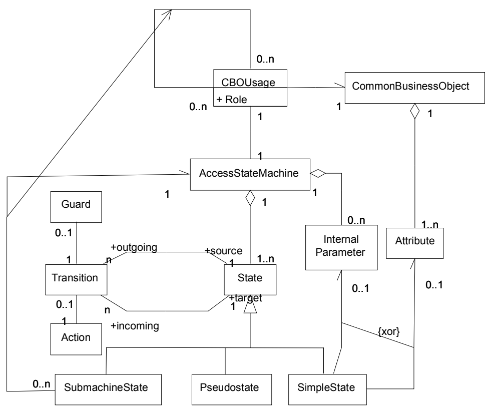


Figure 4. State machine showing the Access to CBOs



```
!ELEMENT DIRXML2 (AccessStateMachine+)>
<!ELEMENT AccessStateMachine (CBOUsage, Path)>
<!ATTLIST AccessStateMachine ID ID #IMPLIED>
<!ELEMENT CBOUsage (#PCDATA)>
<!ELEMENT Path (State, Transition+)>
<!ATTLIST Path ID ID #IMPLIED>
<!ELEMENT State (#PCDATA)>
<!ATTLIST State
   type (A | E | S | P) #IMPLIED>
   submachine IDREF #IMPLIED
   hostelement CDATA #IMPLIED>
<!ELEMENT Transition (Path | EndState | NextPath)>
<!ATTLIST Transition
   access (OK | FAIL) #IMPLIED
   comp (equal | notequal) #IMPLIED
   value CDATA #IMPLIED
   boolean (AND | OR) #IMPLIED
   inparameter CDATA #IMPLIED
   incomp (equal | notequal | greater | less | grequ | lequ) #IMPLIED
   invalue CDATA #IMPLIED
   outparameter CDATA #IMPLIED
   action (set | inc | dec) #IMPLIED
   setvalue CDATA #IMPLIED>
<!ELEMENT EndState EMPTY>
<!ATTLIST EndState success (OK | FAIL) #REQUIRED>
<!ELEMENT NextPath EMPTY>
<!ATTLIST NextPath continue IDREF #REQUIRED>
```

```
<AccessStateMachine ID="ASM003">
  <CBOUsage>Party</CBOUsage>
  <Path>
    <State type="E">TypeOfPartyID</State>
    <Transition comp="equal" value="SellerAssigned">
      <Path>
        <State type="E">PartyID</State>
        <Transition access="OK">
          <EndState success="OK" />
        </Transition>
        <Transition access="FAIL">
          <NextPath continue="P001"/>
        </Transition>
      </Path>
    </Transition>
    <Transition access="FAIL" comp="notequal" value="SellerAssigned">
      <Path ID="P001">
        <State type="S" submachine="ASM005">BillTo</State>
        <Transition access="FAIL">
          <EndState success="FAIL" />
        </Transition>
        <Transition access="OK">
          <Path ID="P002">
            <State type="S" submachine="ASM006">ShipTo</State>
            <Transition access="OK">
              <NextPath continue="P002"/>
            </Transition>
            ...
          </Path>
        </Transition>
      </Path>
    </Transition>
  </Path>
</AccessStateMachine>
```

Figure 5. <<DIR>>XML^2: Meta model, DTD and example

dependency between the relationship of a submachine state and the state machine on the one hand and the recursive relationship between *CBOUsage*s on the other hand, because a submachine can only be used if the corresponding *CBOUsage* is also included in the *CBOUsage* the supermachine belongs to.

Since the state machine definitions should be stored in a registry together with the definitions of the business document types, they have to be represented in a machine-readable format. This format ideally should be in XML itself. Inasmuch as the state machine is represented in UML, XMI would be a candidate format. Nevertheless, due to its expressiveness, XMI would produce large overheads of data not needed for our approach. Thus, we prefer a simple DTD which actually meets our requirements. We have developed a special DTD, which we refer to as <<DIR>>XML^2 - Document Interface Rules for XML Documents in XML. The middle column of Figure 5 shows this DTD. The DTD includes multiple state machines. Each state machine has a unique ID in order to be referenced as submachine. A state machine consists of a *CBOUsage*, stating the name of the corresponding CBO, and a path. A path is always a state with its outgoing transitions. For each state the *type* attribute identifies whether the state accesses an XML attribute, an element, a submachine or an internal parameter. If the access is to a submachine, the *ID* of this state machine is referenced. If an attribute is accessed, the element including the attribute is stated in the *hostelement* attribute.

Each transition can be characterized by multiple attributes: The *access* attribute is used to condition whether the access in the state was successful or not. The *comp* and the *value* attributes control whether the access was equal or not equal to a given value. The *inparameter*, *incomp,* and *invalue* attributes are used to compare an internal parameter to a certain value. To specify a Boolean expression between the guards regarding the access and those regarding the internal parameters, the *boolean* attribute is used. Actions can be defined in the *outparameter*, *action* and *setvalue* attributes. Each transition would logically lead to a new state, but in our DTD it is a choice of three implementations. Firstly, it can lead to a new path (which includes further access states and transitions). Secondly, it can lead to an end state, which either signifies either success or failure. Last, it can lead to a path, which is not included as subelement, but which is referenced by a unique *ID*. This concept is needed to express that a state might have multiple incoming transitions.

The right column of Figure 5 gives an extract of an <<DIR>>XML^2 instance defining the access to buyer information according to our example. It is equivalent to the state machine presented on the right side of Figure 4. In a registry the <<DIR>>XML^2 document instance should accompany a purchase order document DTD or schema.

4. SUMMARY

In this paper we emphasize on the need for additional access rules for generic document structures usually used in XML-based B2B vocabularies. These **D**ocument **I**nterface **R**ules to **XML** documents are defined by adapted **<<**UML**>>** state machines that are expressed by an **XML** DTD as well. Thus, we call this language <<DIR>>XML^2. This language has just served successfully for the case studies in our test field. It has to be applied in a real-business environment to verify whether additional types of constraints are necessary or not.

5. REFERENCES

[1] Coathup, P. Electronic Data Interchange. Computer Bulletin, (1988), 15 - 17

[2] Commerce One. XML Common Business Library. http://www.xcbl.org

[3] Eisenberg, B., and Nickull, D. ebXML Technical Architecture Spec. v1.0.4., (2001). http://www.ebxml.org

[4] Huemer, C. Defining Electronic Data Interchange Transactions with UML. Proceedings of HICSS-34 (Hawaii, January 2001), IEEE Coputer Society Press

[5] ISO. The Open-edi Reference Model. ISO/IEC JTC1/SC30 IS 14662, (1996)

[6] Raman, D. XML/EDI - Cyber Assisted Business in Practice. TIE Holding NV, 1999

[7] UML Revision Task Force. OMG Unified Modeling Language Specification, version 1.3. document ad/99-06-08. Object Management Group (1999)