Tailoring Information to the Needs of Clients

Peter Rittgen
Technical University Darmstadt, Darmstadt, Germany

rittgen@bwl.tu-darmstadt.de

Abstract

The vision of providing clients with information that is tailored to their needs has sparked off a tremendous interest in languages that are on the one hand standardized and hence widely applicable but which are on the other hand also highly flexible and can hence be customized to specific applications. The universal exchange format XML (eXtended Markup Language) is a candidate for such a language but we argue that it does not go far in solving the problems of integrating information from different sources and being provided or used by different actors especially across organizational boundaries. We therefore show existing approaches of enriching XML with application-specific semantics and argue why these are not applicable in many cases. This leads us to introduce a process-oriented method for effectively informing clients on the basis of XML by tailoring documents to their specific needs.

Keywords: Electronic Business, Event-Method Chain, XML Common Business Library, commerce XML, information model

Introduction

Few technologies have inspired so much enthusiasm, in both the academic and the professional world, as the universal data exchange format XML. New languages based on XML or XML tools are introduced on a weekly basis. It is beyond dispute that XML will play an important role in many organizations in the near future. This development is fuelled by the rapidly spreading use of XML for defining exchange formats in e-business and the increasing number of cheap or even free-of-charge tools that support creating, managing and processing XML documents including, for example, concepts and tools that allow for a tight integration of XML with HTML and Java.

To provide for a systematic and effective use of XML we first have to identify potential areas of application. XML was developed to support the exchange of information between informing systems. If we look at a single organization we already find that different systems are used by groups (e.g. departments) or even individuals within the organization. But these systems do not exist in complete isolation: they support sub-processes that eventually contribute to achieving a common goal. To do so they have to share information.

Within one organization we might think about the 'ideal solution', i.e. rebuilding these informing systems so that they all share a common information model, e.g. an object model (Frank, 1998). This would provide us with a high semantical level of integration and the applications could then simply communicate these objects directly without the need of conversion to another format. This could be done e.g. on the basis of the common object request broker architecture CORBA. But in many situations this alternative is not available due to budget restrictions etc. In these cases we might want to "integrate" the systems on a lower semantical level by exchanging well-defined documents between them. But even then we need a “common understanding” of these documents.

An even more compelling need for exchanging documents arises when we consider the fact that today fewer and fewer organizations operate on their own and more and more enterprises engage in building virtual networks to do electronic business. Building a monolithic system for informing the whole network of participants might well prove impossible, so integration has to be achieved via documents.

But how can this be done? Different informing systems typically manage data in a different way: they use different formats (syntax) and they often attribute different meanings to similar or identical items (e.g. fields) or use the same field for different concepts (semantics). To exchange information in the form of documents, the participants have to agree on

1. a common format (document type) and, more importantly, on
2. a common understanding of a class of documents, i.e. a reference semantics.

XML is about to become the de-facto standard for the first issue. But the second issue is typically neglected in the XML literature. Approaches aiming in that direction, i.e. adding application semantics to XML, are discussed in section 2. We argue that these lack important features, for example: abstraction, flexibility and the support of “client-specific” documents. So in order to define reasonable document types and to instantiate them with the appropriate data we have to take into account the processes generating and processing the information contained in the documents. How this can be done is described in section 3.

XML and E-Business

XML has ist roots in the Standard Generalized Markup Language, or SGML, which was developed by Goldfarb, Mosher and Lorie at the IBM Laboratories during the 70s (Goldfarb, & Rubinsky, 1990). The objective behind the development of SGML was a device and system-independent language for describing the logical structure of a document, or more specific a document class or type. SGML is a metalanguage, that is, a means of formally describing a language, in this case, a markup language. Historically, the word markup has been used to describe annotation or other marks within a text intended to instruct a compositor or typist how a particular passage should be printed or laid out such as wavy underlining to indicate boldface, special symbols for passages to be omitted or printed in a particular font and so forth.

XML is a subset of SGML. A detailed description can be found in (Goldfarb, & Prescod, 2000). XML differs other markup languages (such as HTML) in at least 3 respects:

1. the markup is descriptive (i.e. contains no code);
2. the concept of a document type;
3. system independence.

A descriptive markup system uses markup codes which simply provide names to categorize parts of a document. Markup codes such as <para> or \end{list} simply identify a portion of a document and assert of it that “the following item is a paragraph,” or “this is the end of the most recently begun list,” etc. Secondly, XML introduces the notion of a document type, and hence a document type definition (DTD). Documents are regarded as having types, just as other data processed by computers do. The type of a document is formally defined by its constituent parts and their structure. The definition of a report, for example, might be that it consisted of a title and possibly an author, followed by an abstract and a sequence of one or more paragraphs. Anything lacking a title, according to this formal definition, would not formally be a report.

If documents are of known types, a parser can be used to process a document claiming to be of a particular type and check that all the elements required for that document type are indeed present and correctly ordered. More significantly, different documents of the same type can be processed in a uniform way. Programs can be written which take advantage of the knowledge encapsulated in the document structure information, and which can thus incorporate a richer functionality. A DTD for a breakfast menu might, for example, look like this:

```
<!ELEMENT breakfast-menu ( food+ ) >
<!ELEMENT food ( name, price, description, calories ) >
<!ELEMENT name ( #PCDATA ) >
<!ELEMENT price ( #PCDATA ) >
<!ELEMENT description ( #PCDATA ) >
<!ELEMENT calories ( #PCDATA ) >
```

It declares that a breakfast menu consists of several food items (hence the +). Each food element contains the element's name, price description and calories. These are also called the attributes of food because they are of the basic type #PCDATA (string), i.e. they are not subdivided further. Regard that XML has only one simple type so there is no way of preventing "hundred" from appearing as a calorie value instead of "100". An example document of this type, i.e. an actual menu conforming to this DTD might be:

```
<?xml version="1.0" ?>
<breakfast-menu>
  <food>
      <name>Belgian Waffles</name>
      <price>$5.95</price>
      <description>two of our famous Belgian Waffles with plenty of
         real maple syrup</description>
      <calories>650</calories>
  </food>
  ...
</breakfast-menu>
```

Well-formedness, that is correctness of the syntactical structure, and validity, i.e. conformance to the DTD, can be checked automatically by a parser. The following XML document is not well-formed:

```
<food>
      <name>Belgian Waffles</food>
      <price>$5.95
 </name>
```

because the scopes of food and name cross and the *price* field is not terminated. The next example is well-formed but it does not conform to the DTD above and is hence not valid because the value of the price is not of basic type:

```
<food>
  <text> Belgian Waffles </text>
  <price> <dollars> 5.95 </dollars> <euros> 6.20 </euros> </price>
</food>
```

Ever since XML was introduced as a universal data exchange format many organizations have tried to build upon it an infrastructure for the exchange of application-specific documents. Two important approaches investing XML with business-oriented semantics are cXML by Ariba and xCBL by CommerceOne. We are going to sketch them here briefly. For a more detailed account we refer the reader to (Frank, 2000).

cXML (commerce XML) aims at product catalogs and orders. Catalogs come in a static and a dynamic form both with the same content but the latter supporting navigation through the products with a conventional WWW browser. cXML is specified as a set of XML document types which allows for a syntactical check by one of the numerous XML tools. But the semantics has to be validated by specific cXML tools. Existing standards for product details are used such as the product codes and measurement units defined by the United Nations and the ISO Language Code. The details of cXML can be found under http://www.cXML.org/. cXML is documented in the User Guide (Ariba, 2000) which currently defines 23 document types such as Contract, OrderRequest and OrderResponse accompanied by examples. It is not supported by a graphical notation so the reader is forced to deal with XML code to understand the semantics of the document types. As an example for a cXML document we show the particulars of a purchase order:

```
<Contract effectiveDate="2000-01-03T18:39:09-08:00"
          expirationDate="2000-07-03T18:39:09-08:00">
      <SupplierID domain="InternalSupplierID">29</SupplierID>
      <ItemSegment segmentKey=Plant12>
            <ContractItem>
                  <ItemID>
                        <SupplierPartID>pn12345</SupplierPartID>
                  </ItemID>
                  <UnitPrice>
                        <Money currency=USD>40.00</Money>
                  </UnitPrice>
            </ContractItem>
            …
      </ItemSegment>
</Contract>
```

cXML is a proprietary approach but the tools are (at the moment) easily accessible and cheap. Unfortunately it is limited to specific documents (catalogues and orders) only. And although these certainly represent an important class of documents they only account for a small percentage of existing document types. With other, less frequently used, types standardization is less attractive. Here a different approach is more promising (see section 3). cXML also severely restricts the definition of a document's semantics because -

as in XML - the only elementary data type is the string. Numbers, dates, currencies etc. are all encoded as strings and hence threaten to violate the integrity of the informing systems of buyer and seller. Moreover, it is not possible to add features to the documents that are only relevant to a small number of organizations.

xCBL (XML Common Business Library, (Commerce One, 2000)) is a more ambitious effort that not only aims at data exchange but also at supporting E-business applications. Consequently xCBL offers a richer semantics than cXML by introducing an additional layer between XML and xCBL, the so-called schema language SOX. A schema language is a metagrammar for defining the syntactical structure and partial semantics of XML document types. This can be used to partially incorporate semantical features into a language like XML. Two major schema languages for XML are SOX and XML Schema.

SOX (Schema for object-oriented XML) has been specified by Commerce One. It can be found under http://www.w3.org/TR/NOTE-SOX/. It is defined as an XML DTD and it enriches XML with "object-oriented" features (although the important feature of encapsulation was not considered). Nevertheless, SOX represents a notable improvement over XML because SOX allows for the specification of schemata. Just as an XML document has to conform to its DTD, a SOX document has to adhere to its schema. But while XML only provides a check for syntactic validity, a SOX document can also be verified with regard to certain semantical aspects because the set of predefined data types can be augmented by user-defined data types. Moreover, a schema can be specialized from an existing one. Finally, SOX also supports a kind of polymorphism because you can use an instance of a subtype of an element where the corresponding schema requires the supertype. SOX has been submitted to the W3 consortium but the emerging standard, the XML Schema Language, is likely to divert from SOX. In this case CommerceOne plans a new issue of xCBL based on XML Schema.

To summarize, SOX extends the language of DTDs by supporting:

- An extensive (and extensible) set of datatypes,
- Inheritance among element types,
- Namespaces,
- Polymorphic content,
- Embedded documentation and
- Features to enable robust distributed schema management.

All of these features are supported with strong type-checking and validation. A SOX schema is also a valid XML instance according to the SOX DTD, enabling the application of XML content management tools to schema management. The XML Schema Language aims at the same target as SOX but without considering object-oriented features.

xCBL itself is defined as a set of SOX schemas. It contains 55 data types, 11 document type elements (PurchaseOrder, PurchaseOrderResponse, PriceCatalog, ProductCatalog etc.) and more than 300 elements on subordinate levels such as "sections" which correspond to parts of a document (e.g. OrderHeader). The library can be extended by new elements. Beyond the definition of common concepts CommerceOne also provides for an integration on instance level required by certain interorganizational business processes. This is done via "agencies" which manage unique keys for certain product types. It ensures that all participating business partners use the keys in the same way.

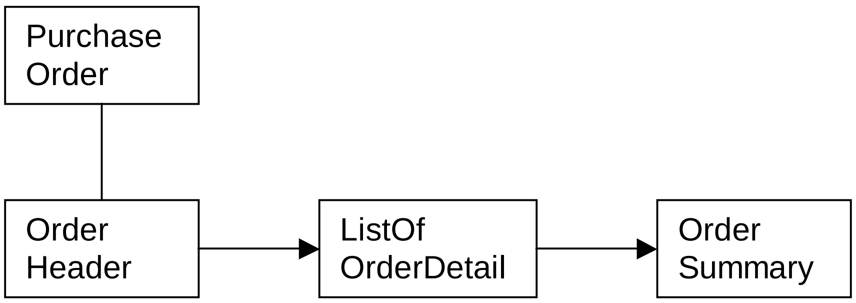

Fig. 1: Structure of a purchase order in xCBL

Document types in xCBL are described with the help of a graphical representation (see Fig. 1) and also in the form of corresponding XML code annotated by explanatory text in natural language. So for an order detail we have e.g.:

Repeating element OrderDetail

OrderDetail	Information about a line item in an order.
BaseItemDetail	General information about the line item.
LineItemNum	*int* The line number on which the item appears in the order.
SubLineItemNum (optional)	*int* Further identifies the item’s position within the order.
SupplierPartNum (optional)	The supplier’s part number for the item.
PartNum	
Agency	
@AgencyID	*AgencyCode*: the agency that assigned the part number. For a list of agency names, see AgencyCode on page 153. If the agency that assigned the part number is not included in this list, specify an AgencyID of "Other" and use the @AgencyOther attribute to specify the agency’s actual name.
PartID	*string* The unique identifier for the part.
PartIDExt (optional)	*string* The part number extension.
BuyerPartNum (optional)	The buyer’s part number for the item.

…

Compared to cXML, xCBL offers significant advantages. The higher semantical level due to SOX improves maintenance and integrity of the library. This is supported by organizational measures such as a certified registration database. This ensures a certain level of quality of proposed and accepted extensions to the library.

On the negative side the participating organizations put their investments in xCBL infrastructure at risk if the new standard for schema languages differs significantly from SOX.

Informing Clients with XML

The applicability of both cXML and xCBL is limited to the exchange of predefined goods for money between medium to large organizations. The documents are specified to such detail that no flexibility remains for small to medium organizations which are typically highly specialized and hence require documents that are likewise specialized to their specific needs. Moreover, neither approach supports transactions other than the standard buy/sell type. But many organizations engage into other forms of cooperation. For example, two companies might agree that the one develops products and the other produces them (and yet a third might market and distribute them). The documents that have to be exchanged here (blue prints, production schedules etc.) are frequently beyond standardization. In addition, approaches such as cXML and xCBL can never handle documents found in project-like settings, i.e. in a one-time cooperation launched for a specific purpose.

What we need in these cases is an approach to design documents according to the specific situation. The following sections elaborate this idea.

Event-driven Method Chain

Information can only be interpreted sensibly in the context in which it is used. Processes are both the source and the sink of information and thus provide information with meaning. An approach for aiming at client-specific information must therefore also address and model the processes dealing with the information. For this purpose we suggest the use of the so-called Event-driven Method Chain (EMC) described in (Rittgen, 2000). This process modeling language is based on the Event-driven Process Chain (EPC) of ARIS (Scheer, 1999) which is typically used by many consultants and large IT departments. Hence a process language based on EPC has a better chance of being accepted by the practitioner than some completely new artefact. The EMC incorporates information objects (generalized to classes) together with their attributes, services provided by a class of objects (also called methods) and resources required for the execution of these services. Execution of a method is triggered by an event. The syntax of an EMC is shown in Fig. 2.

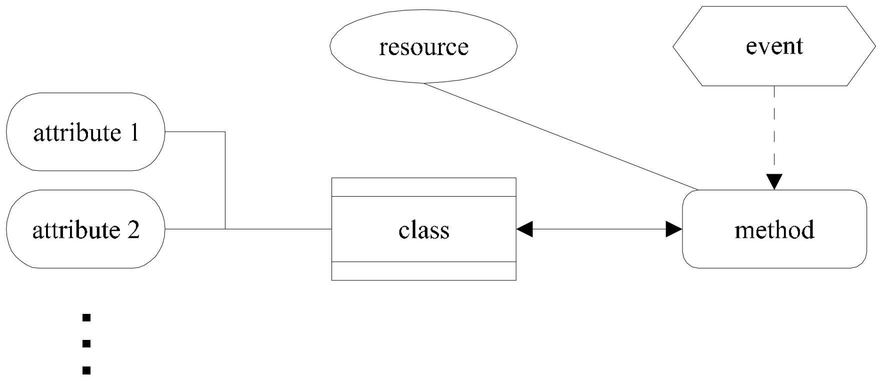

Fig. 2: General syntax of an EMC

As an example for the use of the EMC, Fig. 3 shows a part of the processing of an order within some web application. We assume that the process is fully automated, i.e. it requires no external resources. Upon the arrival of an order, it has to be entered into the system. This service is provided by the class *order* but it involves also the class *customer* because the respective customer has to be recorded in the order. Note: the fact that the service *enter order* is provided by *order* and not by *customer* is not represented in the EMC. But this information, more generally any assignment of services to classes, is vital for the following design phase and hence should be expressed in the information model (see Fig. 4). According to the EMC, the attributes of *order* are *order id* (e.g. a number) and *items* (a list of ordered items and quantities). These attributes also constitute the skeleton of the respective class definition in the information model (see Fig. 4).

After entering the order, it is checked for validity. The outcome of this check is represented by the occurrence of either the event “*order OK*” or the event “*order not OK*”. The XOR split denotes that these events are mutually exclusive. In the case of an invalid order, a refusal of the order is generated and sent via email. The items of a valid order, i.e. the software packages ordered by the customer, are delivered (e.g. via ftp) and the bill is prepared. After this, further processing may occur. Note that no attributes are specified for the class *refusal*. The reason might be that the modeler could not think of appropriate attributes and hence left this to the later stage of developing the information model.

On the basis of this EMC, an initial information model can be specified without further thinking: it simply consists of all classes and their respective attributes as found in the EMC. After this, the following steps yield a complete information model:

1. *assigning services to objects*: from the classes involved in a service, the one providing it has to be se-

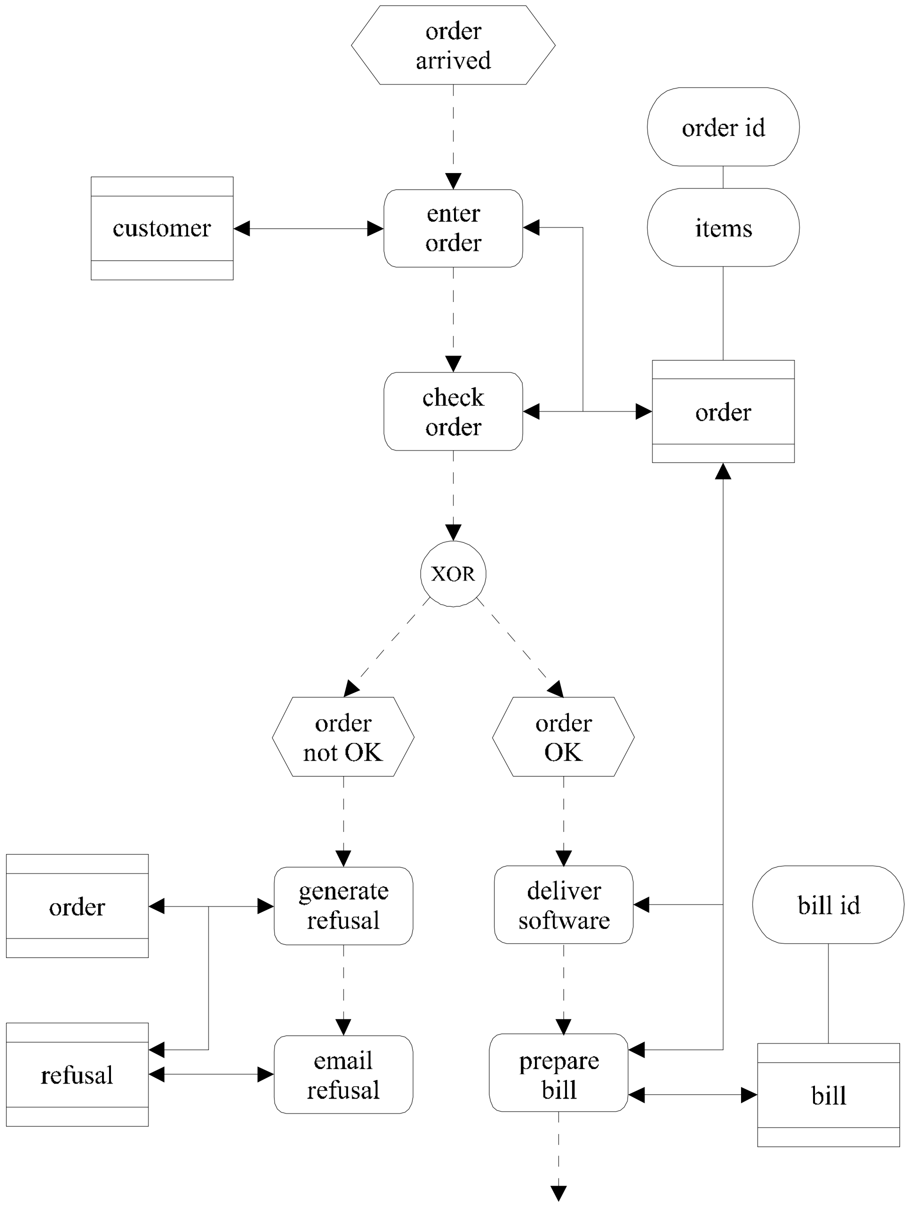

Fig. 3: An example EMC for a web application

lected. There the service is recorded.

2. *finding missing attributes*: each class is thoroughly examined to check whether attributes have been forgotten e.g. because they are not necessary in the context of the current EMC. This step is best performed after all EMCs for the application lie before us.
3. *identifying potentials for generalization*: classes sharing common attributes or services are potential candidates for generalization inheriting these attributes from a common super-class.
4. *establishing associations between classes*: if more than one class is involved in providing a service, there is usually an association between the involved classes.

Starting with the initial information model for the EMC of Fig. 3, we assign the services to classes as indicated in Fig. 4. The EMC gave no attributes for *refusal*. Looking at actual orders, bills and refusals stored in our file cabinet, we find that a refusal contains some explanatory *text* and that all letters carry a *date*. We update the classes accordingly (step 2), and we generalize them to the super-class *document* with attribute *date* (step 3). In the last step, we discover that *customer* and *order* are involved in *enter order*, which leads us to establish an association *places* between them, where a customer can place arbitrarily many orders (0..n) but an order is placed by exactly one customer (1..1). The black triangle indicates the

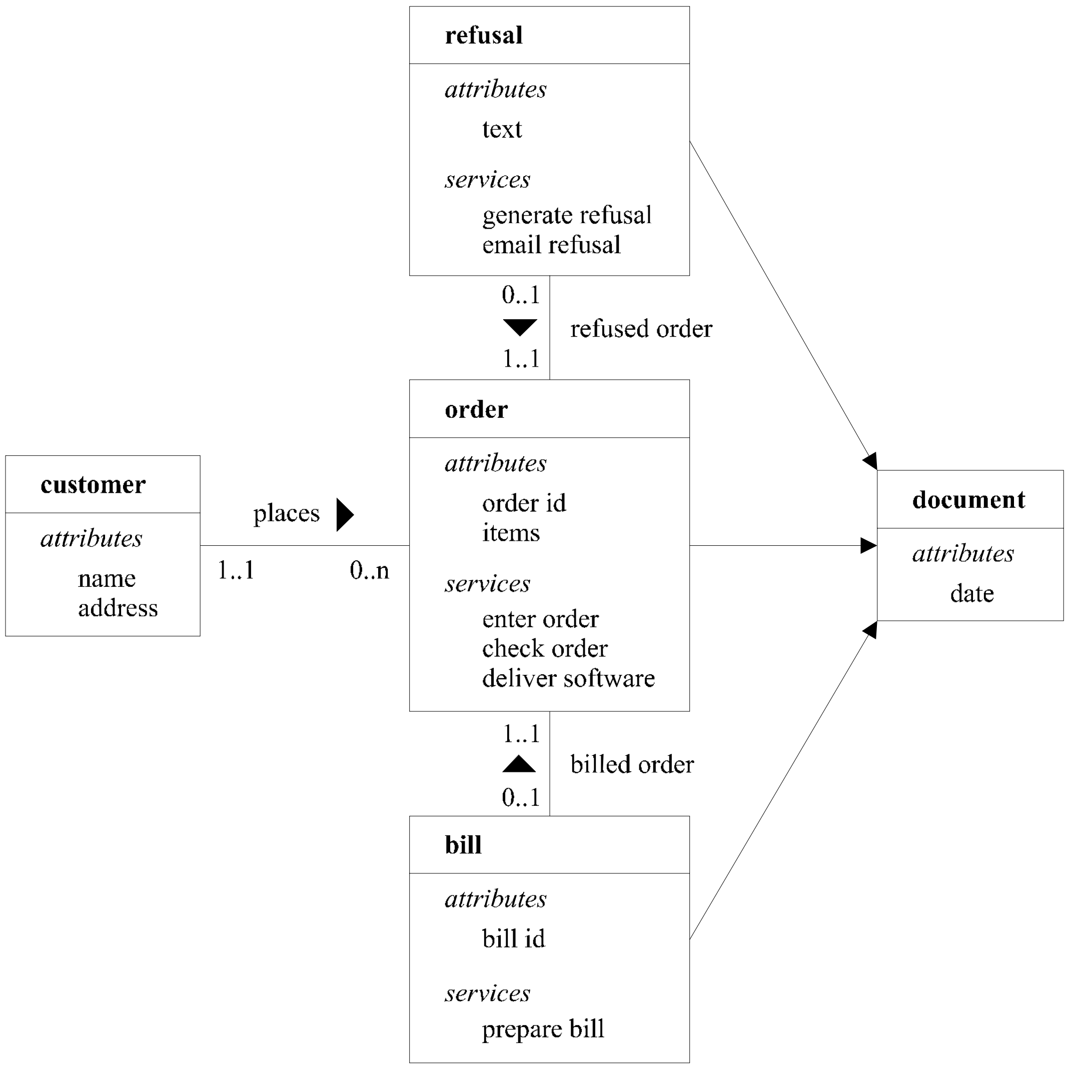

Fig. 4: Information model (class diagram) for the example

reading direction: *customer places order*. In a similar way, *bill* and *refusal* are connected to *order*.

Please observe that the redundancy of having some of the information present in both EMC and information model (e.g. classes, attributes and services) helps to check inter-model consistency and thus serves model integration. The resulting information model is then an ideal starting point for the definition of document types in XML.

Example

The following example illustrates the process described so far. It shows how the DTD templates can be generated for a specific application. Imagine two companies in a supplier-customer relation with frequent and large transactions. They both decide to handle the future exchange of documents electronically to speed up the delivery and payment processes, to cut on administrative and inventory costs and to have both constant and up-to-date information for inventory control and other controlling purposes. Because appropriate tools are readily available they choose XML as an exchange format.

In this scenario a typical procedure would be to list all required documents and to specify them with a conceptual modeling language such as ERM (Entity-Relationship Model, (Chen, 1979)). This ERM would then serve directly as the basis for defining the DTDs. But employing this approach there is no way to decide whether all information required by the business processes of both partners is present and, most of all, if each attribute/field is understood in the same way by the people or information systems responsible for the execution of the individual activities. Common ambiguities include:

- origin and destination of information/data
- intended use
- format of data
- meaning of blank fields
- etc.

To prevent such ambiguities we have to know where the information is produced and where it is used, i.e. the processes generating and "consuming" it. So we have to develop a process model first. Fig. 5 shows a part of an example process in EMC notation. It depicts the order processing on both sides, the supplier

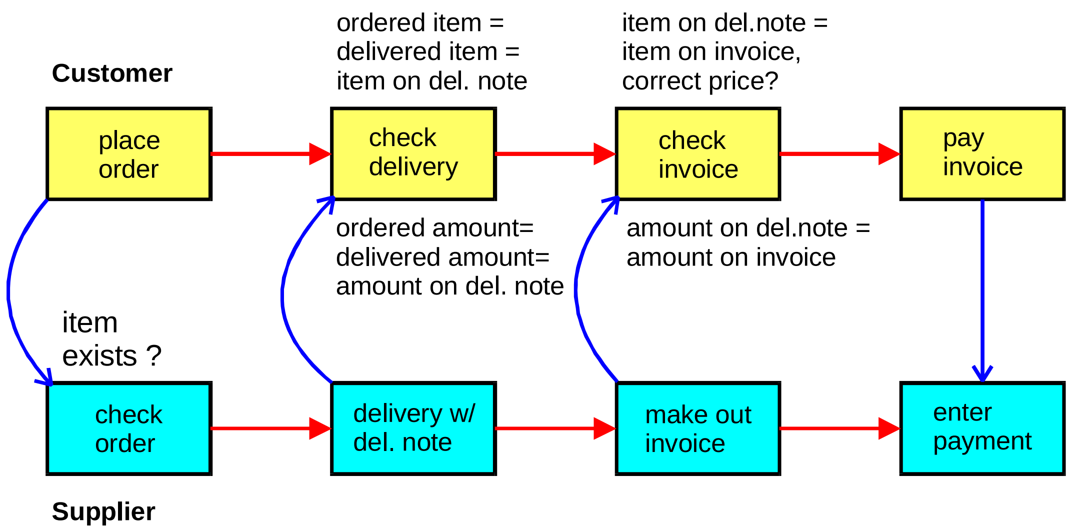
Customer
ordered item = delivered item = item on del. note
item on del.note = item on invoice, correct price?
place order
check delivery
check invoice
pay invoice
ordered amount= delivered amount= amount on del. note
amount on del.note = amount on invoice
item exists ?
check order
delivery w/ del. note
make out invoice
enter payment
Supplier

Fig. 5: Interorganizational business process

side (lower chain) and the customer side (upper chain).

Now, if we want to design the XML documents exchanged between customer and supplier we need to know more about the subprocesses: what exactly does “check delivery” mean? If we ask the manager of the purchase department, he will tell us that this entails two sub-subprocesses: first comparing the delivery note against the delivery (which is done by the warehouseman), and then comparing the delivery note with the actual order by a purchase clerk (see Fig. 6, left side). An interview with the warehouseman reveals that the first process is further subdivided as indicated on the right side of Fig. 6.

If we go on detailing the remaining subprocesses we will finally identify all information required by the subprocesses and hence we can derive the structure of the documents that are exchanged. Fig. 5 shows the relevant information revealed by the detailed modeling.

Now, the information present in the “annotated” EMC of Fig. 5 represents a guideline for designing the DTDs for all involved XML documents. For example, for the validity check of the order we have to assess whether the ordered items belong to the range of supplied products. For that we need the item number and/or its description on the order. Now, to verify the delivery against the order we proceed in two steps as suggested by Fig. 6 because the warehouseman has no access to orders and the clerk from purchasing never sees the actual delivery. Hence we need an intermediate document that accompanies the delivery and is passed on to the purchase department, the delivery note. In order to check the delivery against the note, we compare the items and amounts on the note (which consequently must be listed there) with the ones actually delivered. To verify that all delivered items have indeed been ordered and that the amounts are correct (step two), the same information belongs on the order. In addition, the fact that we have to compare the delivery note against the order also tells us that the delivery note has to refer to the

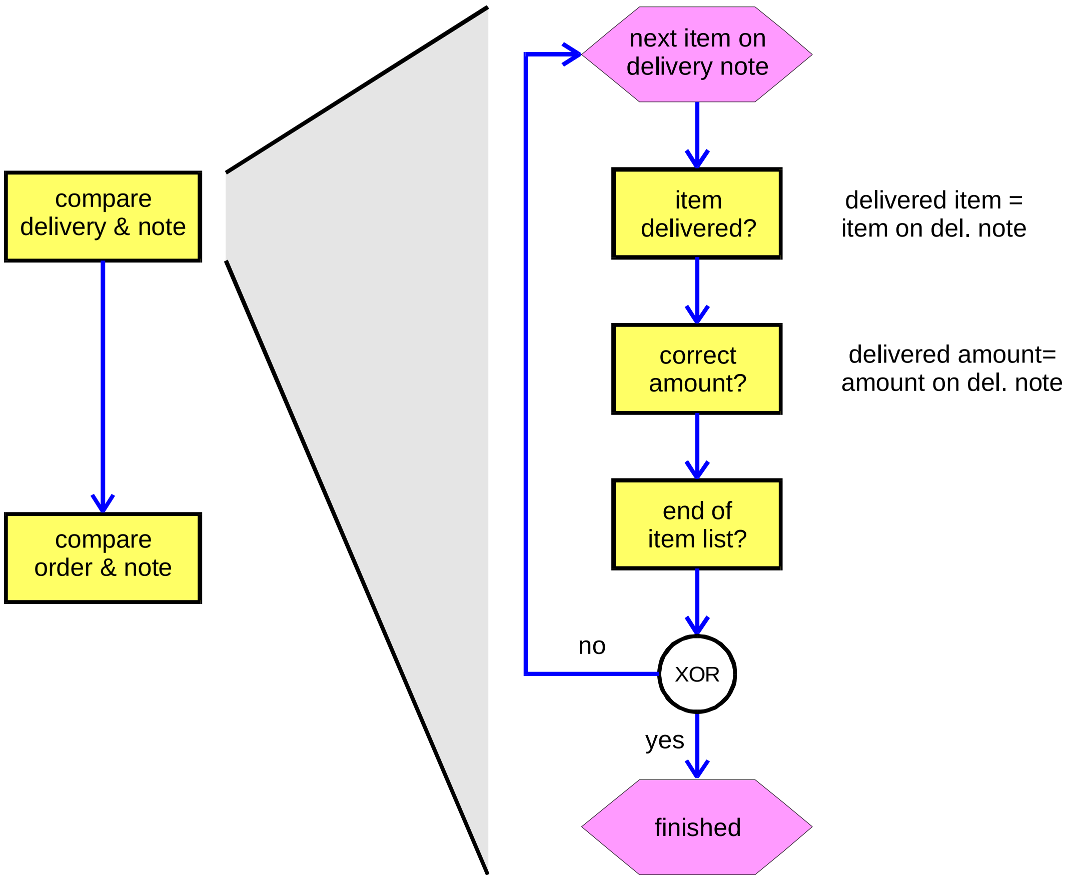

Fig. 6: Subprocess “check delivery”

corresponding order, e.g. by containing the order number.

Putting these pieces together we arrive at the DTDs shown in Fig. 7. The same is done to design the DTD for the invoice and other possible documents involved in the ordering process.

Now, modeling all involved processes in detail may seem a lot of work to do when we just have in mind to design the structure of exchanged documents. But apart from the apparent advantages in building well-structured and flexible documents, the modeling gives us additional benefits: it points at potentially inefficient processes and helps us to determine how to improve these processes. The models guide us in reengineering the current processes and documents and in adapting the organization and the IT infrastructure accordingly.

For example, we might ask why we need the delivery note at all in an EDI setting. It was originally motivated by the fact that the warehouseman does not have access to IT infrastructure. A revised version of this process might not only remove the delivery note and its exchange but also the necessity of a second check in the purchase department.

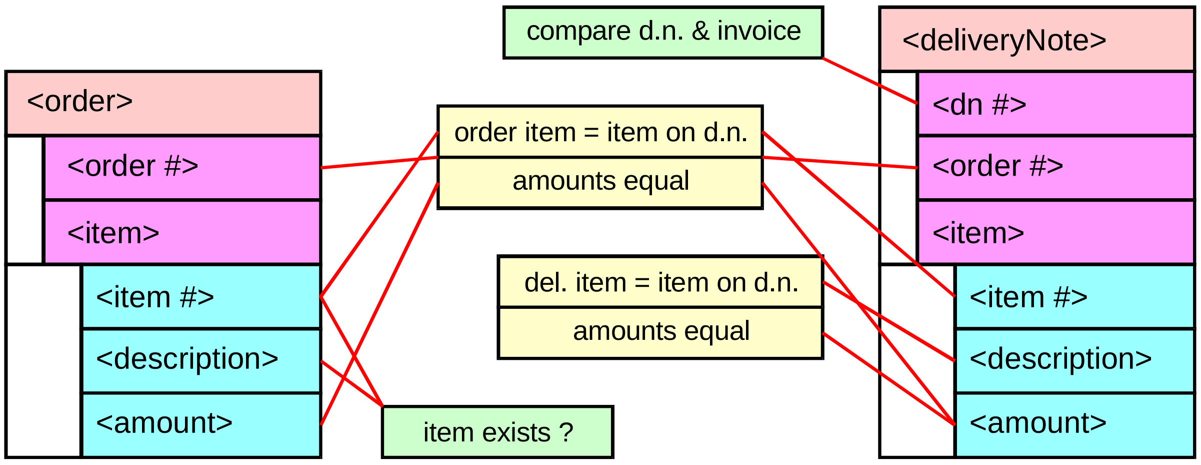

Fig. 7: DTDs for the example process

Conclusion

We started with the assumption that XML plays an increasingly important role as a standard in exchanging documents both within an organization and between organizations. Many providers of e-business infrastructure such as Ariba and CommerceOne rely on it as a basis for defining electronic documents such as catalogs. But the structure of these documents is rigid and cannot be adapted to the needs of a specific organization. Moreover only standard buy-sell transactions are supported. We argue that structure and content of a document depend heavily on the way this document is used, i.e. on the providers and clients of this information. As core processes vary from organization to organization so do the relevant documents. We therefore suggest that the definition of the document types is done after, and based on, the modeling of the relevant informing processes. In order to achieve this aim we introduced a method to describe informing processes as an EMC annotated by details of the information required for each step of the process. Such a model can then be transformed semi-automatically into an information model and from there into a set of DTDs. This approach makes sure that all necessary information is incorporated in some document and that the document's structure is tailored specifically to the needs of the process to be supported.

References

Ariba (2000). cXML User's Guide. Version 1.1. Retrieved February 19, 2002 from the World Wide Web http://www.cXML.org.

Chen, P.P. (1979). The Entity-Relationship Model - Toward a Unified View of Data. In Chu, W.W., & Chen, P.P.: Tutorial: Centralized and Distributed Data Base Systems, 1st International Conference on Distributed Computing Systems, October 1-4, 1979, Huntsville, Alabama. Long Beach, CA: IEEE Computer Society, pp. 166-193.

CommerceOne (2000): Common Business Library. Version 2.0.1, Retrieved February 19, 2002 from the World Wide Web http://www.commerceOne .com/xml/.

Frank, U. (1998). The Memo Object Modelling Language (MEMO-OML), Technical Report 10, University Koblenz-Landau.

Frank, U. (2000). Vergleichende Betrachtung von Standardisierungsvorhaben zur Realisierung von Infrastrukturen für das E-Business (Comparison of Standardization Efforts Towards E-Business Infrastructures). Technical Report 22, University Koblenz-Landau.

Goldfarb, C.F., & Prescod, P. (2000). The XML Handbook. 2nd edition. Upper Saddle River, NJ: Prentice Hall.

Goldfarb, C., & Rubinsky, Y. (1990). The SGML Handbook. Oxford: Clarendon Press.

Rittgen, P. (2002): E-Commerce Software: From Analysis to Design. In Gangopadhyay, A.: Managing Business with Electronic Commerce: Issues & Trends. Hershey, PA: Idea Group, pp. 17-36.

Scheer, A.-W. (1999). *ARIS - Business Process Modeling*. Berlin: Springer.

Biography

Peter Rittgen studied Applied Computer Science and Computational Linguistics at University Koblenz-Landau and graduated with an MSc. After 3 years in industry he worked as Research Assistant in Information Systems at Frankfurt University and received a PhD in Business Administration and Economics with a dissertation on "Process Theory of Scheduling". In 1997 he joined University Koblenz-Landau as Assistant Professor where he did research on Enterprise Modeling and Software Engineering. Currently he works as Associate Professor at Technical University Darmstadt.