Application of Information Technology ■

Plug-and-Play XML:

A Health Care Perspective

Ralf Schweiger, PhD, Simon Hoelzer, MD, Udo Altmann, MD, Joerg Rieger, Joachim Dudeck, MD

Abstract The application of XML (Extensible Markup Language) is still costly. The authors present an approach to ease the development of XML applications. They have developed a Web-based framework that combines existing XML resources into a comprehensive XML application. The XML framework is model-driven, i.e., the authors primarily design XML document models (XML schema, document type definition), and users can enter, search, and view related XML documents using a Web browser. The XML model itself is flexible and might be composed of existing model standards. The second part of the paper relates the approach of the authors to some problems frequently encountered in the clinical documentation process.

■ **J Am Med Inform Assoc.** 2002;9:37–48.

Three problems of the clinical documentation process motivated our work. The intention of this paper is to outline the possible contribution of XML (Extensible Markup Language) to the solution of these problems and to address some implementation issues.

Why Use XML for Clinical Documentation

The first problem is the lack of explicit structure in health-care–related resources such as reports, guidelines, and scientific publications. The exploitation of such resources by electronic means is limited by the fact that many documents are still textual narrations[1] without an explicit structure. One possible exploitation, for example, is the quick access by a health care professional to information of interest. A physician may not want to read a complete clinical guideline if he or she is interested only in a specific part of the guideline.

Irrelevant search results can be reduced to a minimum as soon as we insert meaningful structures such as diagnoses and therapies into clinical documents. XML provides a standard means to describe the structure of a document explicitly[2,3] and to identify meaningful elements in textual narrations.[2]

The second problem is the flexibility of a document's structure. Health care experts agree that the transition from unstructured textual data to structured and coded data will be a migration process.[4] As a consequence, clinical documentation must be flexible in terms of free-textual descriptions, different structural levels, and even individual structures. The structure must not restrict the content. The Clinical Document Architecture (CDA) of Health Level 7 (HL7), for example, defines different structural levels for clinical content.[5] Version 1.0 of the CDA, for example, primarily provides a document header, whereas the document body is rather undefined.

Frequently, physicians prefer narrative text to inflexible forms. Standard forms, on the other hand, can control the quality of clinical documentation and guide a less experienced health care professional in the documentation process. We need a combination of both, standards and flexibility.

We want to stress that XML does not replace the modeling process, which will remain a business of health care professionals and standardization bodies. But XML can help with implementation issues, i.e., when we need to derive hierarchic structures such as documents, messages, or even smaller units from these data models. XML provides a standard means to mix text and structure in various ways and even combine independent structures into more compre-

Affiliation of the authors: Justus-Liebig-University Giessen, Germany.

Correspondence and reprint requests: Ralf Schweiger, PhD, Heinrich-Buff-Ring 44, 35392 Giessen, Germany; e-mail: <ralf.schweiger@informatik.med.uni-giessen.de>.

Received for publication: 5/9/01; accepted for publication: 8/14/01.

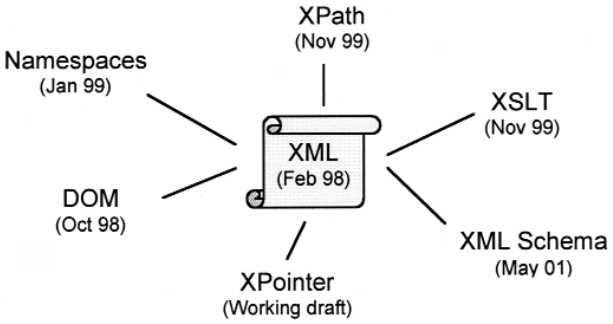

Figure 1 Recommendations of the World Wide Web Consortium (W3C) related to XML. DOM indicates Document Object Model; XPath, XML Path Language; XSLT, Extensible Stylesheet Language for Transformations; XPointer, XML Pointer Language.

hensive document models.[3,6] XML can provide a sort of "composition technology" for such a flexible bottom-up approach.

The third problem is the interface problem. Clinical documents, such as discharge letters, are often summaries of already existing data. However, we often need to develop interfaces for the integration of the data. On the other hand, more and more information systems will use XML as an interchange format, because XML messages can be easily validated against a standard message schema[3] and transformed into a local representation.[7] XML will, consequently, contribute to the reduction of the interface problem.

The "XML Problem"

Figure 1 is a summary of the XML-related standards (World Wide Web Consortium [W3C] recommendations) that we currently use for the development of XML applications.

The Document Object Model (DOM) is a language-neutral programming interface that allows programs to access and update the content and structure of XML documents.[8] XML namespaces[6] provide a method for qualifying element and attribute names used in XML documents, such as <Pathology:Finding> and <Lab:Finding>, in which the namespace prefix (Pathology, Lab) is associated with a URI (uniform resource identifier) reference that points to a document type definition (DTD), XML schema, or other description of the XML namespace.

The XML Path Language (XPath) and the XML Pointer Language (XPointer) allow the description of document fragments.[9] Given the XML structure <patient><name>Jones</name><disease>cancer</disease></patient>, the XPath "/patient[count(disease)>1]/name" would select the names of the patients with more than one disease. XPath can be used for several purposes, such as the transformation and the query of XML documents.

The Extensible Stylesheet Language for Transformations (XSLT) is a language for transforming XML documents into other XML documents.[7] XSLT might facilitate the transformation of XML documents into a human-readable form (XHTML) or the transformation of local XML documents into standard XML messages.

The XML schema allows the structure of an XML document to be described, much like a DTD, and the validation of documents against this schema.[3] What is the XML problem? The term "problem" is probably not right. In fact, XML provides substantial support in the development of XML applications. However, we underestimated the cost of XML application development, assuming that we could develop XML applications very quickly using XML schemas, the DOM and XSLT. But this was not the case. It is relatively easy, for example, to parse and transform XML documents using the DOM and XSLT. The user entry of XML documents, on the other hand, is still a problem. Existing XML editors do usually not run inside a Web browser, and users have to deal with XML markup. In addition, we need a search engine that supports the retrieval of XML documents by means of a given content. We consequently have to compose the given "XML ingredients" into more comprehensive application services.

Methods

The development of XML applications required more time than we expected, and we felt that it could be facilitated. Each XML application needs a user interface for the entry and rendition of XML documents, a message system for the integration of other XML resources, and a storage system for the persistence of XML documents. We therefore developed a Web-based application framework that makes existing XML components cooperate to provide this generic application functionality.

Web-based XML Application Framework

Figure 2 shows the core architecture of the Web-based XML application framework—specifically, the software components and relationships between them. The core architecture of the framework is a result of dividing an XML application into a user interface (x-applet), a storage system (x-servlet), and

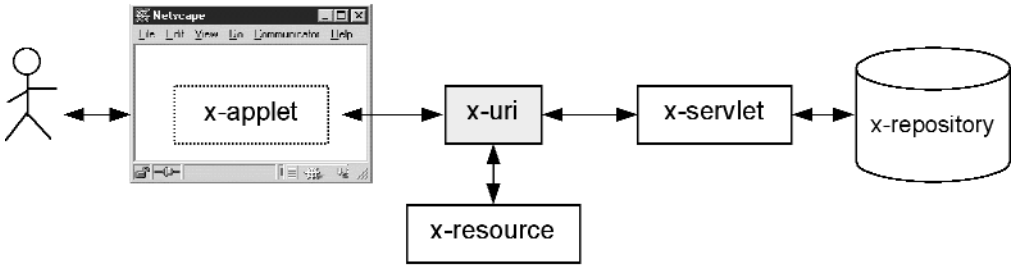

Figure 2 Web-based XML application framework.

a message system (x-uri). The XML framework provides the glue between existing XML components and makes them communicate, to provide in turn a set of generic application services. The XML framework defines a set of "hot spots" that need to be changed to integrate new XML components.

Each component of the XML framework uses existing XML tools to perform a specific task. The *x-repository* stores and relates XML resources such as XML schemas, DTDs, XSL/CSS style sheets and XML documents. The *x-servlet* provides a simple application programming interface for access to the x-repository, i.e., functions for the storage and retrieval of XML resources. The x-servlet hides the implementation details of the x-repository (indexing methods, resource distribution, etc.) and makes the framework independent of the underlying storage system (file system, XML database). The *x-uri* component is the glue of the whole system and connects different "XML-speaking" Internet resources by a standard URL connection. The x-uri component manages, for example, the communication between the x-applet and the x-servlet.

An *x-resource* (abbreviation for XML-related resource) can be static, such as an XML schema, a CSS style sheet, or a DTD; or it can be dynamic, such as another application that receives and sends XML structured data. The x-applet and the x-servlet are dynamic x-resources. The *x-applet* runs in the Web browser and provides the user with a dynamic and flexible user interface for the management of XML resources. The user interface comprises functions for the entry, rendition, and query of XML documents.

The Model-and-Apply Philosophy

The overall goal of the Web-based XML application framework is to ease the development of XML applications. Basing the XML framework on a Web platform is one aspect of ease of application, because Web applications can be developed and deployed very quickly. The central idea, however, was to implement our vision of XML, as described above. We wanted to reduce the cost of developing an XML application to the cost of creating a DTD/XML schema and a number of related CSS/XSLT stylesheets. In the simplest case, we would need to create a DTD or XML schema to provide an authoring system that could be deployed over the Web.

Figure 3 shows how the XML framework works. We feed the XML framework with an XML model that is subdivided into a structural model and an interface model. *Structural model* is a generic term for a DTD or XML schema. *Interface model* describes "the behavior of a document type" to the user and to other application systems. The interface model may comprise, for example, several CSS/XSL style sheets to provide several user views of a given document type. Furthermore, we can establish communication lines

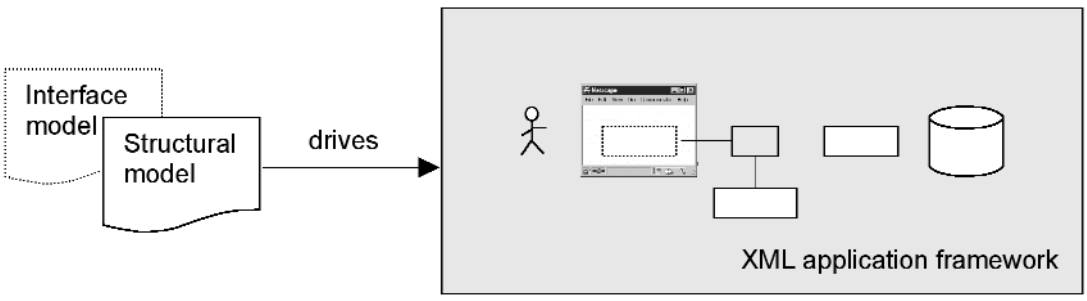

Figure 3 Model-and-apply philosophy.

```
<!ELEMENT Notes (Patient, Note*)>
<!ELEMENT Patient (Name, Birth, Gender)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Birth (#PCDATA)>
<!ELEMENT Gender (#PCDATA)>
<!ELEMENT Note (#PCDATA| History|
    RIM0105c:Observation| Diagnosis|
RIM0105c:Diet| Medication| RIM0105c:Procedure)*>
<!ELEMENT History (#PCDATA)>
<!ELEMENT RIM0105c:Observation (#PCDATA)>
<!ELEMENT Diagnosis (#PCDATA)>
<!ELEMENT RIM0105c:Diet (#PCDATA)>
<!ELEMENT Medication (#PCDATA)>
  <!ELEMENT RIM0105c:Procedure (#PCDATA)>
```

Figure 4 Structural model represented as document type definition (DTD).

to independent application systems, specify communication triggers, and provide XSLT style sheets that describe necessary transformations of the XML data. The interface model is optional, because we provide a default interface. We consequently use the term *XML model* as a generic term for all XML resources that are related to a given DTD or XML schema. The XML model needs to be stored in the x-repository of our XML framework (see Figure 2).

We illustrate our method by an example, which does not claim to be complete. The intention is to illustrate the main features of the XML application framework.

Let us assume that a physician wants to enter clinical notes in a structured fashion, so that they can be accessed by such meaningful concepts as diagnoses and therapies. Let us further assume a patient management system with an XML interface.

We first have to define and relate the concepts of interest, such as patient characteristics, diagnoses, and therapies. The result of this modeling activity is the structural model (Figure 4), which is represented as a DTD (see "DTD and XML Schema Compared," below).

According to the structural model, we can associate any number of notes with a given patient. The <Note> element has a mixed content model, i.e., it contains text in which we want to identify further elements such as observations and diagnoses. The <Observation>, <Diet> and <Procedure> elements are prefixed by "RIM0105c" (qualified names), just to indicate that these concepts are drawn from the HL7 Reference Information Model (RIM) version 1.5c. The RIM is a comprehensive source of all information subjects used in any HL7 specification, which makes it possible to share consistent meaning beyond a local context. The DTD example is not complete in the sense that a RIM observation would be composite. In addition, we could also draw the patient concept from the RIM. The example is meant just to show how the XML frame-

Table 1 ■

Extensible Set of XML Attributes Interpreted by the XML Framework.

Attribute	Description
Label	Provide a human readable label for the XML markup. Default is element name.
Form	Define a data entry form for the user. Default is a text field. 1. "text: 10, 70" (text area that is 10 characters high and 70 characters wide, text is unlimited) 2. "menu: ralf/r, simon, udo/u" (menu of three possible values in label/code style where /code is optional, default is code = label) 3. "radio: male/m, female/f" (radio buttons in label/code style with optional /code) 4. "check:" (check box) 5. "URI of code list" (see text under "DTD and XML Schema Compared")
Default	Define a default value for the XML element. Maps onto XML schema's default attribute. The value "javascript: var now = new Date(); return now.toGMTString()" automatically inserts the current date and time on creation of a new document instance.
Connect	The value of this attribute is an extended URI of the form "URI XPath XSLT-1 XSLT-2". The URI works with different protocols (HTTP, IIOP etc.). The URI extensions are optional. The XML framework connects to the Internet location URI on change of the XML nodes identified by XPath. The XML framework sends the XSLT-1 transformed document to the URI. The URI returns an XML result. The XSLT-2 transformed result is presented to the user according to the result's type. Possible result types are <select><element_instance_option/>...</select> and <alert>message</alert>. Examples of use: (1) On change of the patient's medication, we want to connect to a URI that checks the interactions between the drugs and returns an alert if necessary. (2) On change of the patient's name, we want to offer the user an updated list of matching patients (dynamic list). (3) We want to connect to an external vocabulary server that is maintained by a standardization body.

work behaves when we summarize several elements into a common XML namespace (see "The User Interface," below).

We now implement a small interface model. We focus on the interface knowledge that is difficult to express with a CSS/XSL style sheet. Some of this interface knowledge would be accommodated in a so-called XForm. However, the XForms approach is still a W3C working draft,[10] and the software support is limited. As a result, we developed a similar and workable approach. Table 1 defines a small set of XML attributes. The XML framework is able to interpret the values of these XML attributes in order to establish interfaces to the user and to other application systems.

We can now enhance the DTD shown in Figure 4 with interface knowledge by simply attaching the XML framework attributes (shown in Table 1) to the XML elements. The enhanced model is shown in Figure 5. This work needs to be done only once for every DTD or XML schema. The XML framework attributes are prefixed by "xf" to indicate that they belong to the XML framework's namespace.

The *xf:connect* attribute attached to the <Patient> element instructs the XML framework to connect to the patient management system (more exactly to the servlet wrapper of the patient management system) whenever the name of the patient (relative XPath) is changed. The *xf:default* attribute of the <Name> element changes the patient's name from empty to blank and forces the XML framework to connect to the patient management system on creation of a new XML document instance. The patient management system returns a list of matching patient records that is offered to the human user for selection.

Because of the structural model, the <Gender> element might have any textual content. However, the user can enter only the values "m" and "f," i.e., the <Gender> element is actually coded. The question whether coded values should belong to the interface model or the structural model is discussed later (see "DTD and XML Schema Compared," below). If the number of code values were bigger, as for the <Medication> element, we would retrieve the code values along with their human-readable labels from an external vocabulary. The external terminology can be either an XML file (form attribute of Table 1) or a vocabulary server with an XML interface (connect attribute of Table 1) that is maintained by a standardization body.

Let' assume that someone wants to use XML attributes in the structural model:

```
<!ELEMENT Notes (Patient, Note*)>

<!ELEMENT Patient (Name, Birth, Gender)>

<!ATTLIST Patient
      xf:connect CDATA #FIXED
        'http://host/patient-servlet Name'>

<!ELEMENT Name (#PCDATA)>

<!ATTLIST Name
      xf:default CDATA #FIXED ' '>

<!ELEMENT Birth (#PCDATA)>

<!ATTLIST Birth
      xf:label CDATA #FIXED 'Birth date'>

<!ELEMENT Gender (#PCDATA)>

<!ATTLIST Gender
      xf:form CDATA #FIXED 'radio: male/m,
        female/f'>

<!ELEMENT Note (#PCDATA| History|
  RIM0105c:Observation| Diagnosis|
  RIM0105c:Diet| Medication| RIM0105c:Procedure)*>

<!ELEMENT History (#PCDATA)>

<!ELEMENT RIM0105c:Observation (#PCDATA)>

<!ELEMENT Diagnosis (#PCDATA)>

<!ELEMENT RIM0105c:Diet (#PCDATA)>

<!ELEMENT Medication (#PCDATA)>

<!ELEMENT RIM0105c:Procedure (#PCDATA)>
```

Figure 5 XML model, DTD-enhanced by some interface knowledge.

```
<!ATTLIST Diagnosis
  Code CDATA #IMPLIED>
```

In fact, it is difficult to define a label for the Code attribute of the <Diagnosis> element. However, for XML schemas this limitation no longer exists. In addition, the European Committee for Standardization (CEN)[11] recommends mapping information content into XML elements and using XML attributes for the representation of processing and specification data (metadata) only.

The User Interface

We can now store the XML model, i.e., the enhanced DTD of Figure 5, in the x-repository of the XML framework (Figure 2). On selection of an XML model, the XML framework automatically generates a corresponding user interface. The user interface runs in a Web browser and offers functions for the entry, query, and rendition of related XML documents. Each user func-

Table 2 ■

Extensible Set of Application Services Offered to the User

Operator	Description
selModel	Select a model from the x-repository.
newQry	Create an empty query form (empty query is default).
edQry	Edit the current query.
selStyle	Select a CSS/XSL style sheet from the x-repository for document rendition (a default style sheet exists).
newDoc	Create an empty document.
putDoc	Store the current document in the x-repository.
selDoc	Select the documents from the x-repository that comply with the search criteria entered into the query form (see edQry).
viewDoc	Render the current document using the selected CSS/XSL style sheet (see selStyle).
edDoc	Edit the current document.
delDoc	Delete the current document from the x-repository.
newEl	Insert another element of the selected element type.
delEl	Delete the selected element instance.

tion is composed of an *operator* and an *operand*. We currently distinguish five operand types: Model, Document, Query, Style, and Element. *Model* is one of the XML models plugged into our XML framework. *Document* represents an XML storage unit, i.e., a model instance. *Query* is an XML document that contains search relevant data only. Selective query data can be specified at any structural level. *Style* is either a CSS or XSL style sheet associated with the XML model inside the x-repository. Style sheets can provide specific views (restrictive, highlighting) of a given document type. *Element* represents an XML element defined in the structural model. For each operand type, we now provide a number of operators. Table 2 shows an extensible set of core functions currently supported by the XML framework.

Figure 6 shows the document entry form of the user interface. The style of the user interface can be varied by additional CSS/XSL style sheets. Basically, the user interface is organized into three frames. The left frame contains a list of documents that comply with a given query (list frame). The right side is divided into an upper frame with the applicable operators (operator frame) and into a lower frame with the content (content frame). The content frame in Figure 6 contains a document editor and can be switched to a query editor using the *edQry* operation.

As a fundamental function, the user can control the document details. We can click, for example, onto the <Patient> element—more precisely, the element's label—and the user interface hides the element's structure and displays the element's content on the right side of the element's label (see the first note in Figure 6). According to the interface model, the user interface offers a list of patients that is automatically updated when the name of the patient is changed. The radio button in front of an element can be used

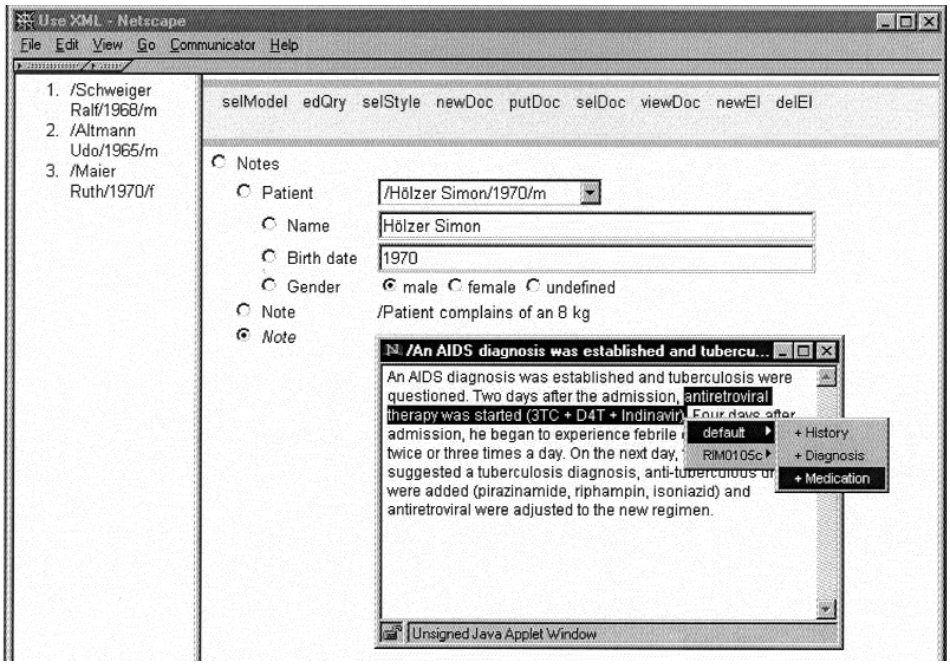

Figure 6 User interface (document entry) generated from the XML model.

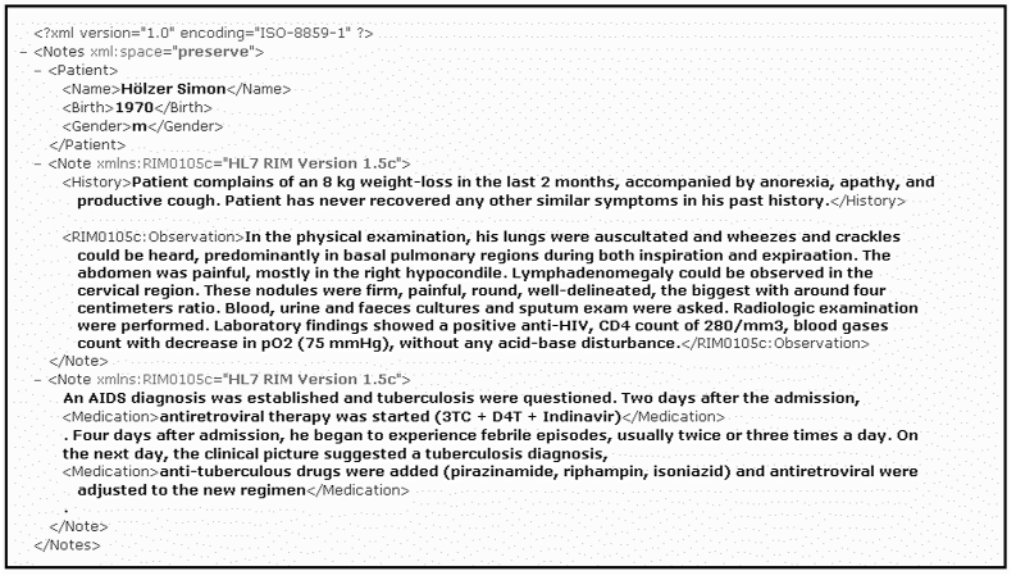

Figure 7 The structured and coded (Gender) result of the user input.

to select a complex element and to get the applicable element operators. The *newEl* operation applies to the selected element (second note in Figure 6) because the <Note> element may repeat according to the structural model shown in Figure 4.

Since the <Note> element has a mixed content model, the user must update the element's content in an XML editor. We developed our own XML editor because existing editors had no programming interface that would allow integration with the XML framework. The XML editor is encapsulated in a separate *x-text* component used by the x-applet (see Figure 2) to handle XML elements with a mixed content model. When the user highlights a text segment, the editor automatically displays a list of those XML namespaces (default namespace and RIM0105c) that offer a "tag" or "untag" action. The user selects a namespace and receives a list with the XML elements that apply in the given context; invalid XML markup does not appear.

The title of the editor window displays the XML context of the highlighted text segment as an XPath expression. In the example shown in Figure 6, we have a single XML text node under the root node. The context might be "not well-formed." In this case the editor does not offer any XML markup for addition. The "+Medication" menu item means that the user can annotate the highlighted text accordingly (add structure). What would happen if the user highlighted a subsegment of the newly added <Medication> element? No namespace would apply, because according to the DTD, the <Medication> element must not have nested elements.

It is also possible to remove structure from the text. A "−Medication" item would display a list with the <Medication> elements that are annotated in the highlighted segment. The user could then untag one or all <Medication> elements. As a basic principle, the user interface avoids all user actions that do not validate against the DTD or XML schema. For example, the user cannot store the document (putDoc) if mandatory elements such as the patient name are missing.

Results

The XML framework has been implemented successfully on the basis of standardized Internet concepts such as URIs and XML-related technologies. During the testing phase of our system, many possible applications became evident. The quick and structured acquisition of clinical notes is only one example. We may use the approach further to model a questionnaire and to carry out an Internet-wide survey quickly. All that is needed on the client side is a Web browser.

Another application field is unstructured documentation. Especially in the health care domain, many data are still entered as plain text. We can use the XML framework to insert some structure in a quick and gradual fashion. Our first opportunities to do this have been with pathology reports[12] and clinical guidelines at our university hospital.

We now return to the problems described in the introduction, to explain the contribution of the XML framework to their solution.

Table 3 ■

Model Flexibility

Before Transformation	After Transformation	Description
Model <A/> Content <A>a</A>	Model <A/><B/> Content <A>a</A><B/>	Extension
Model <A/><B/> Content <A>a</A><B>b</B>	Model <A/> Content <A>a</A>	Deletion
Model <A/><B/> Content <A>a</A><B>b</B>	Model <B/><A/> Content <B>b</B><A>a</A>	Reordering
Model <A/> Content <A>b</A>	Model <A>#<B/>#</A> Content <A><B>b</B></A>	Refinement
Model <A>#<B/>#</A> Content <A><B>b</B></A>	Model <A>#</A> Content <A>b</A>	Simplification

NOTE: The pound sign (#) represents any well-formed XML content.

Document Exploitation

Figure 7 shows the structured result of the user input. On the basis of such XML documents we are able to exploit the textual content for various purposes. If a physician is interested in the symptoms of a given diagnosis, he or she needs only to enter the diagnosis into the query form and activate the "select document" function. The user may enter the search diagnosis at the level of the <Note> element or more specifically at the level of the <Diagnosis> element. The system will return a list with the matching notes, i.e., a list with the XML documents that contain the search diagnosis in the given context (<Note> element, <Diagnosis> element). We can further provide a style sheet that highlights the interesting relationship between <Observation> elements and <Diagnosis> elements. The XML markup consequently forms the basis for the separation of relevant and irrelevant content.

The XML model limits the textual exploitation. As a rule, the more structure (DTD, XML schema) and code values (form attribute in Table 1) the XML model enforces, the better the potential exploitation. In addition, some responsibility remains to the application developer and the user. For example, let's assume that the user searches for an ICD code inside the <Diagnosis> element, using the query form. The XML framework will find all documents with the XML structure <Notes>#<Note>#<Diagnosis>#ICD#</Diagnosis>#</Note>#</Notes>, where # represents any well-formed XML content. The XML framework will not find the documents in which the ICD code occurs in the <Note> element but not in a <Diagnosis> context. Consequently, it is up to the user to interpret the search result in the correct way if the <Diagnosis> element is optional and other users do not annotate their diagnoses. The application developer could provide, for example, a more verbose element label (Table 1) to indicate the option of the <Diagnosis> element.

The XML mixed content model is very interesting from the health care perspective, because clinical documents are often narrative in content or contain some narrative sections (discharge letters, pathology reports etc.). The automatic and reliable extraction of clinical elements such as diagnoses and therapies from textual narrations is still a problem.

Model Flexibility

Flexibility in this context refers to the model design; that is, we may change existing XML models without needing to change related XML documents. The XML framework automatically modulates the XML application according to the latest model design. We illustrate this by the <Notes> example. If we are interested in the address of the patient, we simply need to add an <Address> element to the <Patient> element shown in Figure 5. The <Address> element might be composed or not. The user interface immediately reflects the model extension. Table 3 is a summary of the currently supported model transformations.

How can old XML documents remain compatible with the new XML model? The key point is that the DTD or XML schema does not necessarily describe the storage format of an XML document but rather the overall form of organization. The XML framework has the ability to reorganize XML documents on the fly. What happens if the model is changed? The storage format of an XML document remains unchanged for the time being. If the user wants to

operate on that document, the XML framework reorganizes the document according to the latest model design. The user can make the reorganized document persistent using the putDoc function.

In Table 3, the column "After Transformation" shows that the content of an XML document does not really change unless we delete model elements. However, the structure of the content changes. The first three model transformations are easy to understand. For refinement purposes, the content model of element A has to be changed into a mixed content model, that is, the content of A is annotated in the XML editor (Figure 6). The XML editor offers the additional element B for textual annotation. If the model is simplified, the XML editor just ignores markup that is not declared in the DTD or XML schema. The textual content, however, is preserved.

Ease of Integration

Ease of integration is a result of our extended URI concept. An extended URI has the following form: URI XPath XSLT XSLT. The URI extensions are optional. The extended URI instructs the XML framework to connect to the Internet location URI on change of the XML nodes identified by XPath (communication trigger). The XML framework transforms the current XML document using the first XSLT style sheet and sends the transformation result, such as an HL7 message, to the URI. The URI returns an XML message to the XML framework, and the XML framework transforms the message using the second XSLT style sheet. The XML framework can now present the communication result to the user according to the result's type (alert, selection, etc.). In the <Notes> example, the extended URI—

```
<!ELEMENT Patient (Name, Birth, Gender)>
<!ATTLIST Patient
    xf:connect CDATA #FIXED
      'http://host/patient-servlet Name'>
```

—instructs the XML framework to connect to the patient-servlet whenever the <Name> of the patient (relative XPath) changes. The patient-servlet returns a list with the patient records that match the given document data. The patient records need to have the structure declared in the DTD, since no XSLT style sheet has been specified. The result's type is a selection, in this case a selection of patient records; that is, the XML framework dynamically generates a menu that allows the user to select a patient record from the list. The extended URI works with different protocols, such as HTTP, FTP and IIOP, which makes the communication concept quite flexible.

Discussion

Many XML-related standards and software components have been developed since February 1998, when XML became a W3C recommendation. However, comprehensive XML applications that support the whole document life cycle are still lacking. At the XML Europe 2000 conference, the situation of XML was described as follows[13]:

> …however exciting the prospect of a 'Semantic Web' is, it won't gain broad support until tools are available for the widespread user creation and manipulation of XML.

Currently, XML is a developer's tool kit rather than a user's application, and Web browsers are just beginning to support XML-related standards. As a result, we developed a Web-based XML application framework (Figure 2) that makes existing XML components "speak" to each other in order to provide the user with generic application services (Table 2). The overall goal of the XML framework is to ease the development of XML applications. The central concept of the XML framework is referred to as "model and apply" (Figure 3); that is, the development of XML applications is reduced to the provision of a set of related XML resources (DTDs, XML schemas, CSS/XSL style sheets). We refer to a set of related XML resources as an XML model. A major design concept of the XML framework was the flexibility of such XML models (Table 3).

Current Limitations of the XML Framework

The functionality of the XML framework is not yet complete. For example, the XML framework does not allow a user to electronically sign an XML document. The paramount goal of our implementation efforts was to prove the feasibility of our approach. Because of the modular and integrative design of the XML framework, however, the application developer can extend its core functionality. Despite its limitations, the XML framework might serve as a workable starting point, because it already provides basic application services for the entry, persistence, retrieval, and rendition of XML documents. In addition, the XML framework runs on a Web platform and is therefore easy to deploy. The XML framework might currently add value in those areas where clinical information is entered primarily as narrative text and where we can start with simple XML models.

The XML framework is still under construction. We are currently working on a solution for version 1.0 of the CDA, because we want to establish the CDA as a

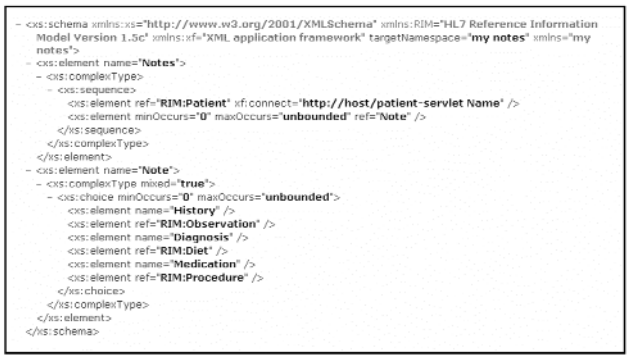

Figure 8 The <Notes> example represented as XML schema.

communication standard between physician offices and hospitals.[14] The CDA provides an exchange model for clinical documents (such as discharge summaries and progress notes) and aims to bring the health care industry closer to the realization of an electronic medical record. We encountered some limitations of our XML framework in this context. For example, the XML framework can not yet handle choices of XML elements outside of an XML mixed content model. Regarding the CDA, sequences of XML elements are prevalent as in most clinical documents. Sporadic choices can occur, however, as the following CDA element shows:

```
<!ELEMENT service_actor (
service_actor.type_cd,
participation_tmr?,
signature_cd?,
(person | organization),
local_header*)>
```

Furthermore, the XML framework currently does not support XML attributes in the DTD or XML schema. Thus far, we have adopted a recommendation of the CEN,[11] according to which an information content is mapped into XML elements. The CDA, however, uses XML attributes for the content.

The choice limitation seems to be much more serious than the attribute limitation with respect to the user interface. The user is interested in model choices but not in XML issues. The XML framework could therefore internally map XML attributes onto more expressive XML elements. (See "User Interface" on next page.)

DTD and XML Schema Compared

Figure 5 shows a simple XML model for clinical notes. The XML model is basically a DTD. Figure 8 shows an equivalent XML schema. In contrast to the DTD approach, we assume that the elements <Patient>, <Observation>, <Diet>, and <Procedure> are defined in a separate RIM schema. The RIM elements might be complex. We can now create our <Notes> schema in a composite way by referring to RIM elements where possible and by defining local extensions where necessary. Again, the model does not claim to be a good model. We want primarily to illustrate the composition principle. The content of the <History> element is not restricted; that is, the <History> element can have any well-formed XML content.

The XML schemas have some strong advantages over DTDs. The strongest is that XML schemas are XML documents, whereas DTDs are not XML documents. In other words, all facilities available for XML documents in general can be used for XML schemas in particular. For example, we could basically use our XML framework to manage a repository of XML schemas. For this purpose, we mainly need to provide a sort of Meta model, that is, a DTD or XML schema that describes an XML schema.

Another advantage results from the fact that XML schemas are XML documents. The DTD limitation that we cannot attach interface information to XML attributes (Figure 5 and related material) no longer exists:

```
<xs:attribute name="code" type="xs:string"
  xf:label="code of diagnosis"/>
```

Furthermore, the expressiveness of XML schemas is superior to the expressiveness of DTDs. The XML schema provides data types such as "anyURI" and "date" that can be used to restrict and validate the content of both XML elements and XML attributes. In addition, XML schemas allow a better reuse of already defined model concepts and provide, therefore, a greater "composite power" than DTDs. The XML schema learned from the DTD entities what kinds of model definitions need to be reusable, such as XML content models and XML attribute groups. Finally, the

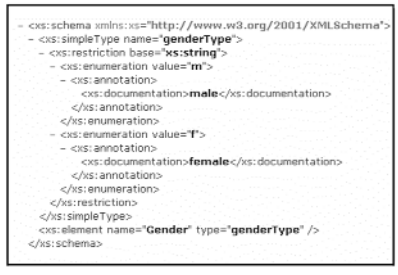

Figure 9 Controlled vocabulary for the <Gender> element.

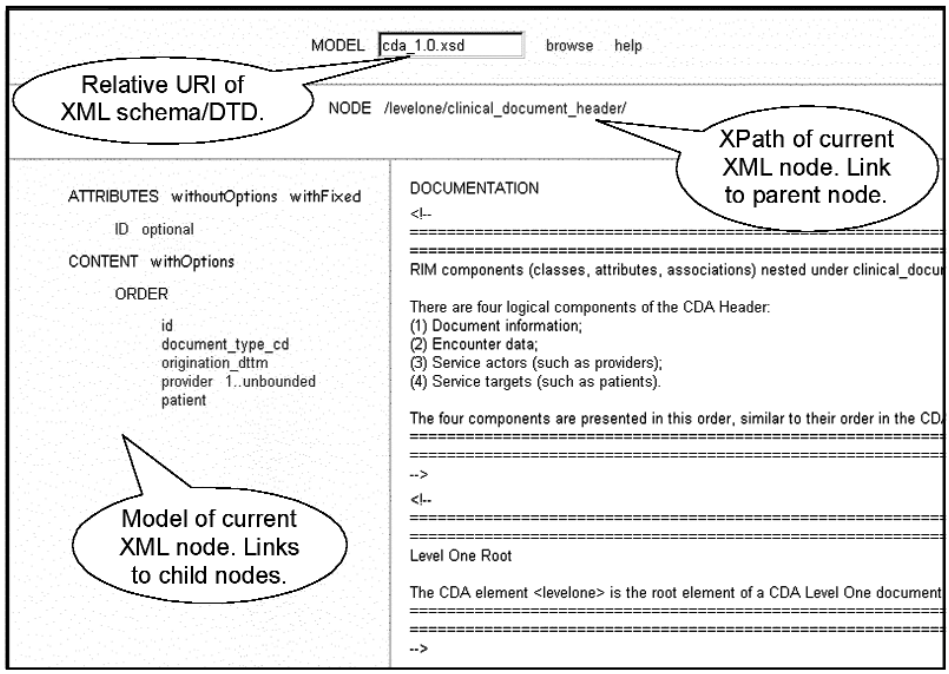

Figure 10 An DTD/XML schema browser illustrating some reduction principles (XPath, options).

XML schema allows documentation elements to be linked to each model concept; that is, XML schemas may contain not only a document model but also a precise documentation of this model.

We can use the XML schema expressiveness to replace local markup, defined in Table 1, by standard XML schema markup. The XML schema already provides a default attribute for XML elements and attributes. We can represent code values as XML schema enumeration values and human-readable labels as XML schema annotations.

Figure 9 shows an XML schema definition of the <Gender> element in Figure 5. Independent standardization bodies can consequently maintain controlled vocabularies in a standardized syntax, and application systems such as the XML framework can easily integrate these vocabularies.

User Interface

The real challenge of the XML framework was and is the user interface. We learned from experience that users do not care about XML. They do not want to mix their text with XML markup, and, as a consequence, they are not interested in XML-specific concerns, such as well-formedness and validity. Users do not care whether their text is represented as XML attribute value or as XML element content. It is the task of the user interface to keep all these XML issues away from the user. Existing XML editors still seem to underestimate this concern.

The XML mixed content model seems to be a very interesting storage model, because it allows a text to be structured and cohesive at the same time. From the user interface perspective, however, it is a challenging concept if we take into account what has been said before. Figure 6 illustrates our context-sensitive XML editor, which is used for the entry of XML mixed content models (<Note> element). It is not yet clear whether users will adopt this approach.

Another concern with respect to the user interface might be the size of the XML schema/DTD. In the CDA context, we are currently investigating an approach to the user interface that might handle large document models. Figure 10 shows a Web component that allows a user to specify any XML schema or DTD and browse the document model. This tool provides full XML schema and DTD support.

The user interface must provide functions that allow a user to reduce a document model to the XML nodes of interest. We can use the XPath language to describe the XML context. Users should also be able to blind out document options and recall them as needed. The lower right frame shown in Figure 10 is currently used to explain the meaning of the selected XML node, but it could be also used for data entry purposes.

Conclusion

Comprehensive XML applications are still lacking. Web browsers are just beginning to support existing XML standards and will not be able to cover all aspects of XML application. The increasing diversity of XML standards and XML components does not take into account that XML application is a matter of bringing them together.

We developed a Web-based XML application framework that makes existing XML resources communicate. The XML framework aims to ease the development of XML applications. Our vision is that we can develop XML applications by pure means of such XML-related standards as XML schemas and CSS/XSL style sheets. The XML framework is a first approach to realization of this vision, but it is limited by the fact that some XML-related work is not yet complete.

We refer to a set of related XML resources as an XML model. On the basis of XML models, the XML framework provides users with generic application services for the entry, query, and rendition of related XML documents. A crucial design principle of the XML framework is the model flexibility that allows the creation and maintenance of XML models in a step-by-step fashion.

If we want to communicate the meaning of XML documents, we must standardize the underlying XML models; in other words, XML modeling and standardization will play a significant role in the future. This task will remain a challenge to standardization bodies. However, XML can support the implementation of a flexible and composite bottom-up approach to this process. The XML schema, for example, suggests the construction of content models that can be shared and used as building blocks for creating new schemas. The role of our XML framework is an operational one—how to realize value from existing XML tool kits and models.

References ■

1. Roberts A. Analyzing XML health records. Proc XML Europe 2000; Jun 12–16, 2000; Paris, France. 2000:377–80.
2. Bray T, Paoli J, Sperberg-McQueen CM (eds). Extensible Markup Language (XML) 1.0. W3C recommendation REC-xml-19980210. Feb 10, 1998. Available at: http://www.w3.org/TR/1998/REC-xml-19980210. Accessed Nov 16, 2001.
3. Thompson HS, Beech D, Maloney M, Mendelsohn N (eds). XML Schema Part 1: Structures. W3C recommendation REC-xmlschema-1-20010502. May 2, 2001. Available at: http://www.w3.org/TR/xmlschema-1/. Accessed Nov 16, 2001.
4. Cimino JJ. Data storage and knowledge representation for clinical workstations. Int J Biomed Comput. 1994;34:185–94.
5. Clinical Document Architecture Framework, release 1.0, Ann Arbor, Mich.: Health Level Seven, Inc., 2000. Available at: www.hl7.org. Publication ANSI/HL7 CDA R1.0-2000.
6. Bray T, Hollander D, Layman A (eds). Namespaces in XML. W3C recommendation REC-xml-names-19990114. Jan 14, 1999. Available at http://www.w3.org/TR/REC-xml-names/. Accessed Nov 16, 2001.
7. Clark J (ed). XSL Transformations (XSLT), version 1.0. W3C recommendation REC-xslt-19991116. Nov 16, 1999. Available at http://www.w3.org/TR/1999/REC-xslt-19991116. Accessed Nov 16, 2001.
8. Apparao V, Byrne S, Champion M, et al. (eds). Document Object Model (DOM) Level 1 Specification, version 1.0. W3C recommendation REC-DOM-Level-1-19981001. Oct 1, 1998. Available at http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001. Accessed Nov 16, 2001.
9. Clark J, De Rose S (eds). XML Path Language (XPath), version 1.0. W3C recommendation REC-xpath-19991116. Nov 16, 1999. Available at http://www.w3.org/TR/1999/REC-xpath-19991116. Accessed Nov 16, 2001.
10. Dubinko M, Dietl J, Merrick R, Raggett D, Raman TV, Welsh LB (eds). XForms 1.0. W3C working draft WD-xforms-20010608. Jun 8, 2001. Available at: http://www.w3.org/TR/2001/WD-xforms-20010608/. Accessed Nov 16, 2001.
11. European Committee for Standardization, Technical Committee for Health Informatics (CEN TC251). Task Force XML—Final Report. Brussels, Belgium: CEN TC251, 1999. Publication CEN TC251/N99-067.
12. Schweiger R, Tafazzoli A, Dudeck J. Using XML for flexible data entry in healthcare, example use for pathology. Proc XML Europe 2000; Jun 12–16, 2000; Paris, France. 2000:357–62.
13. Dumbill E. The state of XML. Proc XML Europe 2000; Jun 12–16, 2000; Paris, France. 2000:33–7.
14. Heitmann KU, Dudeck J. Important step to fill the gap? The German SCIPHOX project. Proceedings of XML Europe 2001 Conference; May 21–25, 2001; Berlin, Germany.