Designing XML Schemas for Bioinformatics

BioTechniques 34:1200-1211 (June 2003)

ABSTRACT

Data interchange bioinformatics databases will, in the future, most likely take place using extensible markup language (XML). The document structure will be described by an XML Schema rather than a document type definition (DTD). To ensure flexibility, the XML Schema must incorporate aspects of Object-Oriented Modeling. This impinges on the choice of the data model, which, in turn, is based on the organization of bioinformatics data by biologists. Thus, there is a need for the general bioinformatics community to be aware of the design issues relating to XML Schema. This paper, which is aimed at a general bioinformatics audience, uses examples to describe the differences between a DTD and an XML Schema and indicates how Unified Modeling Language diagrams may be used to incorporate Object-Oriented Modeling in the design of schema.

INTRODUCTION

The extensible markup language (XML) (1) is rapidly emerging as the universal standard for the transport of data between different bioinformatics databases (http://www.w3.org/XML). In addition, Native XML Databases are being developed to store XML documents and their schema directly in an XML format. Until now, most document schemas in bioinformatics have been based on the document type definition (DTD); they are not written in XML itself. The new W3C standard, XML Schema, is not only XML-based but it also allows for a wide range of data types and is extensible; that is, new data types may be defined. It is a sophisticated means of describing the structure and constraints on the content model of an XML document (www.25hoursaday.com/StoringAndQueryingXML.html). XML Schemas have many advantages over DTDs, and there is a need to transition from the older DTDs into XML Schemas.

Example1.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE seq SYSTEM "Example1.dtd">

<seq id="my_seq" name="NUCLEAR RIBONUCLEOPROTEIN">
  <dbxref>
    <database>SWISS-PROT</database>
    <unique_id>P09651</unique_id>
  </dbxref>
 <residues type="aa"> SKSESPKEPEQLRKLFIGGLSFETTDESLRSHF-
EQWGTLTDCVVMRDPNTKRSRGFGFVTYATVEEVDAAMNARPHKVDGR-
VVEPKRAVSREDSQRPGAHLTVKKIFVGGIKEDTEEHHLRDYFEQYGKI-
EVIEIMTDRGSGKKRGFAFVTFDDHDSVDKIVIQKYHTVNGHNCEVRKALS-
KQEMASASSSQRGRSGSGNFGGG-RGGGFGGNDNFGRGGNFSGRGG-
FGGSRGGGGYGGSGDGYNGFGNDGGYGGGGPGYSGGSRGYGSGGQ-
GYGNQGSGYGGSGSYDSYNNGG-GRGFGGGSGSNFGGGGSYNDFGNY-
NNQSSNFGPMKGGNFGGRSSGPYGGGGQYFAKPRNQGGYGGSSSSS-
SYGSGRRF
 </residues>
</seq>
```

The design of the document schema is critically important, the schema being a separate set of definitions that define the allowed structure of the document and all its variants. An XML Schema is more complex than a simple DTD, and more consideration must be given to design issues. XML Schemas are object-oriented; indeed, the key to the design of flexible and extensible XML Schema is the incorporation of object-oriented design (OOD) principles. The unified modeling language (UML) (2,3) is a graphical-based scheme that aids in the design process.

There are several reasons why biologists should concern themselves with what some would view as a low-level implementation issue. First, bioinformatics is more about the design, transmission, storage, and processing of existing data than it is about generating new data. Biologists may find themselves losing control of their own subject if they are unable to participate in data management at some level. Second, data design is part of the larger information design problem (e.g., semantics and ontology construction). Finally, and more technically, XML Schema design determines the kind of queries that can be made of an XML document, and thus the kind of information that can be obtained from it—an issue of more than passing interest to a biologist. This paper aims to make biologists aware of the basics of data modeling and outlines how UML can be used as an aid for understanding and designing XML Schemas using examples taken from bioinformatics.

MATERIALS AND METHODS

An Example DTD and XML Schema

When XML was released in 1998, XML Schemas were not yet available, and the document structure was specified by DTDs, which were a part of the original standardized generalized markup language (SGML). XML may be viewed as a smaller, simpler subset of SGML, designed specifically for data interchange on the World Wide Web. The essential feature of both markup languages is that the information governing the content (here, the XML source document), the structure (the DTD or XML Schema), and the presentation [perhaps a cascading style sheet (CSS)] are stored in separate files.

The following bioinformatics XML document, Example1.xml, (www.industry.ebi.ac.uk/~alan/XMLWorkshop) uses a (external) DTD called Example1.dtd. The DTD, which is really a set of rules to which the document must conform, may also be placed internally

Example1.dtd

```
<!ELEMENT seq (dbxref*, residues?) >
<!ATTLIST seq              id        ID        #REQUIRED
                           name      CDATA     #IMPLIED
                           length    CDATA     #IMPLIED >
<!ELEMENT dbxref (database, unique_id) >
<!ELEMENT database (#PCDATA)>
<!ELEMENT unique_id (#PCDATA)>

<!ELEMENT residues (#PCDATA)>
<!ATTLIST residues  type  (dna | rna | aa)  #REQUIRED>
```

Example1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="seq" type="seqType"/>

<xsd:complexType name="seqType">
  <xsd:sequence>
    <xsd:element name="dbxref" type="dbxrefType"/>
    <xsd:element ref="residues" minOccurs="1"/>
  </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string"/>
    <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="dbxrefType">
  <xsd:sequence>
   <xsd:element name="database" type="xsd:string"/>
   <xsd:element name="unique_id" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="residues">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="type" type="residuesType"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>

<xsd:simpleType name="residuesType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="dna"/>
    <xsd:enumeration value="rna"/>
    <xsd:enumeration value="aa"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

within the XML file.

From Example1, we see that: the root element "seq" contains zero or more of elements "dbxref", followed by zero or one of element "residues"; element "seq" has attributes "id", "name", and "length"; element "dbxref" contains exactly one "database" element, followed by exactly one "unique_id" element; elements "database", "unique_id", and "residues" are allowed to contain character data; element "residues" must contain an attribute called "type", which must have as a value one of "dna", "rna", or "aa". Any document satisfying these rules is considered to be "valid". Example1.xml may also be described by an XML Schema called Example1.xsd and shown below ("xsd" means "XML Schema document").

An XML Schema incorporates rules in a manner similar to a DTD, but in a way that is more flexible. An XML Schema can define a variety of (complex) data types and give each a unique name, whereas a DTD allows only for data in the form of text. Understanding the concept of a data type is the key to understanding XML Schema. Primitive data types are predefined and consist of types such as "String", or "abcdefg" for example, integer or type "int", which is a whole number like 7, 0, -13, etc., type "char", a character such as "a" or "7", and decimal numbers represented by type "double" for numbers, such as 23.4, etc. The data in an attribute must be a primitive data type or a restriction of it. A simple element is one that has no attributes and also contains no other elements; its content is one of the primitive data types. A complex data type is user-defined and is a particular arrangement of elements, attributes, content, and the allowed primitive data types contained within it. The rules governing the complex data types are specified in the schema. New complex elements may reuse existing data types and even extend them in different ways. This feature introduces the object-oriented concept of inheritance into XML. For example, we may define a complex data type called "database" as given by the instance:

<database ID="1">SWISS-PROT</database>.

A new data type, called "database-date" could be defined by extending

"database" to include the attribute "date":
<database date="3July1998" ID="1">SWISS-PROT</database>.

This is an example of inheritance by extension. Clearly, the XML Schema is more complex than the DTD. A good deal of this is because the XML Schema code is both reusable and extensible. The essential features of an XML Schema may be modeled graphically with no real loss of information using UML. This is a lot easier to learn than XML Schema, as only a small subset of UML is used, and allows a domain expert, such as a biologist, to participate in the schema design.

Example2.xsd

```
<xsd:complexType name="dbxrefType" abstracts="true" >
  <xsd:sequence>
    <xsd:element name="database" type="xsd:string"/>
    <xsd:element name="unique_id" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="dbx2refType">
  <xsd:complexContent>
    <xsd:extension base="dbx2refType">
      <xsd:sequence maxOccurs="unbounded">
      <xsd:element name="database2" type="xsd:string">
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Example2.xml

```
<dbxref>
 <database>SWISS-PROT</database>
  <unique_id>P09651</unique_id>
  <database2>WORLD-GENBANK</database2>
 </dbxref>
```

How XML Schemas Incorporate Object-Oriented Concepts

The three main principles of OOD that need to be incorporated into the design of schemas are (www-106.ibm.com/developerworks/xml/library/x-flex schema): (*i*) encapsulation; (*ii*) inheritance; and (*iii*) polymorphism. Encapsulation means the tight combination of elements, attributes, content, and data types into a single unit, which we will call a class. The class is defined in the schema and includes the rules that determine how the elements and attributes may be combined. The class acts as a template, and each copy made from it is called an object or instance. Instances differ from each other by having different values assigned to the attributes and different arrangements of sub-elements. In XML, the entire schema can function as a class or template. Each XML document is one particular instance of the schema. Example1.xml, is an instance of the XML Schema, Example1.xsd. Within the schema, each complex element, or data type, can also function as a class. A schema can be designed so that it is made up of a collection of complex data types, combined in modular fashion. Modularity is a key feature of an object-oriented system.

The most significant difference between XML Schemas and DTDs is the ability of the XML Schema to define and reuse complex data types. In Example1.xsd, the complex data type "dbxrefType" is defined. It is a complex data type because it contains other elements. In Example1.xsd, "seq", "dbxref", and "residues" are complex; while "database" and "unique_id" are simple. Data types may be defined in a file separately from the schema. Data types that are immediate child elements of <schema> are global elements and may be referenced by other schemas, (www-106.ibm.com/developerworks/xml/library/x-flexschema). In this way, inheritance is incorporated into XML. Types are predefined and may be used elsewhere by simply referencing the type (4). Inheritance in XML Schemas is the ability to adapt an existing data type and extend it by the addition of new elements or attributes. Consider Example2.xsd. The new data type, called "dbx2refType", which inherits from "dbxrefType" and extends it, would describe an element such as Example2.xml. The data type "dbxrefType" is called the super class, and the derived data type "dbx2refType" is its subclass.

Polymorphism means "taking different forms". Polymorphism in XML occurs when several derived types inherit from the same super class, and each one implements some given feature in a different way. Simple inheritance allows different derived data types to add attributes, but with polymorphism, a particular attribute is implemented in a different way for each subclass. For example, one derived type might restrict the attribute "ID" to be of the form, ID = "AB48-3", whereas another derived type might restrict "ID" to be of the form ID = "354-33-CD". These have the same attribute name, "ID", but different simple data type—the essence of polymorphism.

UML and its Application to XML Schemas

XML Schema code is not intuitive

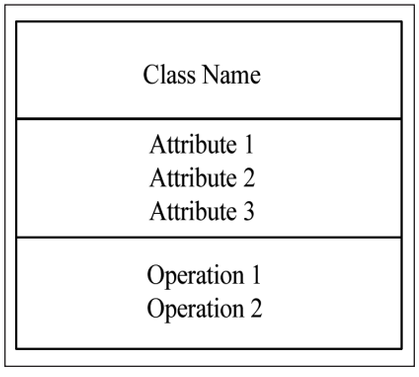

Figure 1. Symbol for a class diagram.

Table 1. PP2 Genes and Proteins (NCBI Data)

Gene	Species	M_r	Amino Acid	GenBank
CbmPP 2[a]	Squash	24.474	219	L31550
CbpPP2	Pumpkin	24.565	219	Z22647
AgPP2-1	Celery	19.849	181	AY114139
AgPP2-2	Celery	19.732	179	AY114140
AtPP2-A1	*Arabidopsis*	28.100	247	At4g19840
AtPP2-A2	*Arabidopsis*	17.914	155	At4g19840

[a]Z17331, L31551, L31552 additional sequence information.

Table 2. PP2-Like Genes in the *Arabidopsis* (TAIR Data)

Gene[a]	Amino Acid[b]	M_r	EST or cDNA[c]
At4g19840	246	28.1	AV557741; R65550; Z26078; A1995445; T41813; AV527977
At4g19850	154	17.9	AV567627; AV518792; AV558075
At1g09155	288	32.5	BE524163

[a]Gene accession numbers.
[b]Number of predicted amino acids in TAIR.
[c]GenBank accession nos. for expressed sequence tag (EST) or cDNA.

Table 3. The PP2-Like Gene Family (NCBI and TAIR Data)

Species	Family	Gene	GenBank
Celery	*Apiaceae*	AgPP2	Table 1
Arabidopsis	*Brassicaceae*	AtPP2	Table 2
Cicer arietinum	*Fabaceae*	CaPP2	A1271666
C. digitata	*Cucurbitaceae*	CbdPP2	Table 1
Squash	*Cucurbitaceae*	CbmPP2	Table 1
Pumpkin	*Cucurbitaceae*	CbpPP2	Table 1
Physcomitrella	*Funariaceae*	PpPP2	B1436506

and is rather difficult to read. On the other hand, UML is a graphical language that allows us to see object-oriented concepts and relationships very clearly. When writing code, a programmer can quickly become immersed in the need to be detailed and precise. A UML model for an XML Schema helps to maintain precision without getting mired in the coding details. UML can be used to visualize relationships and concepts in a manner independent of any implementation language and also gives us an overall view of the system (www.rational.com/media/whitepapers/TP189draft.pdf) and (www.rational.com/media/uml/resources/media/uml_xmlschema33.pdf).

UML class diagrams may be adapted to describe data models (5). The general UML class encapsulates attributes together with the operations that act on the attributes. The symbol for a class of objects is a rectangle, called a UML class icon, as shown in Figure 1. The rectangle is divided into three sections: the name of the class (in capitals) goes in the top section, the attributes (data) of the class go in the middle section, and the operations (behaviors) go in the bottom section (www.w3.org/TR/1998/NOTE-XML-data-0105) (6,7). For applications to XML, the operations are elided (there are no operations in XML), and the lower box is left empty.

Class diagrams consist of a set of UML class icons, together with their relationships, and are used for modeling object-oriented systems such as data models. A UML class diagram may be adapted to model an XML Schema. Figure 2 is a UML class diagram that illustrates inheritance. The complex data type "dbxrefType" is the parent of the child data type "dbx2refType". The derived data type inherits all the elements and attributes of the super data type.

EXAMPLE

Tables 1, 2, and 3 constitute the information that will be marked up as XML. The data on the PP2 genes and proteins are from the huge GenBank® (www.ncbi.nih.gov) and The Arabidopsis Information Resource (TAIR) (www.arabidopsis.org) databases.

The first step in the process is completed at the conceptual level, and it is

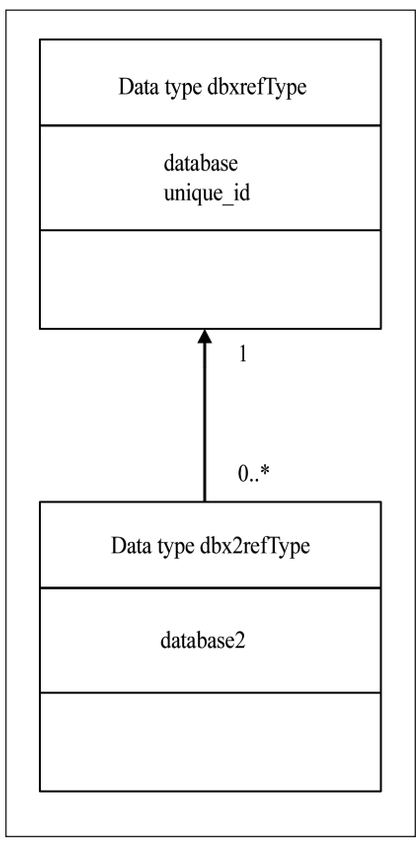

Figure 2. UML diagram showing inheritance in a XML Schema.

Example3.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
        <!—======Root Element Definition==============—>
        <xsd:element name="PP2-LikeProteins" type="PP2-LikeProteinsType">
                <xsd:annotation>
                        <xsd:documentation>Describes the diversity of the Superfamily of Phloem Lectins
            (Phloem Protein2: PP2) in Angiosperms
                    </xsd:documentation>
                 </xsd:annotation>
        </xsd:element>
<!—=====Complex Type Definitions===========—>
        <xsd:complexType name="PP2-LikeProteinsType">
                <xsd:sequence>
                        <xsd:element name="gene_description" type="GeneDescriptionType"/>
                </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="GeneDescriptionType">
                <xsd:sequence>
                        <xsd:element name="gene" type="GeneType"/>
                        <xsd:element name="gene_properties" type="GenePropertiesType"/>
                        <xsd:element name="source" type="SourceType"/>
                        <xsd:element name="species_name" type="SpeciesType"/>
                </xsd:sequence>
        </xsd:complexType>
<!—=====Simple Type Definitions============—>
        <xsd:simpleType name="GeneType">
                <xsd:restriction base="xsd:string">
                        <xsd:pattern value="[A-Z][a-z]\d[a-z]\d{5}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="GenBankType">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="ncbi"/>
                        <xsd:enumeration value="tair"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="AccessionNoType">
                <xsd:restriction base="xsd:string">
                        <xsd:pattern value="[A-Z]{1,2}\d{5,6}"/>
                </xsd:restriction>
        </xsd:simpleType>
</xsd:schema>
```

here that domain experts such as biologists can participate in the design of the data model. Biologists could construct a conceptual-level UML diagram with only a restricted knowledge of XML Schema. We must identify the data entities that will constitute the model and establish the associations between them. It is here that the element and attribute names are chosen.

To identify objects, simply write a list of all of the things in the system. Next, some criterion is needed to distinguish between what will become an element (simple or complex) and what will become an attribute. Complex elements (i.e., data types) are classes, and an object is a specific instance of the class. An object is something that can be distinctly identified; it exists independently of anything else, whereas an attribute does not. Reviewing the tables, we can identify some potential objects: "Gene" (which we will refer to as "GeneDescription"), "Species", "Family", "Molecular Mass", "Amino Acid", "GenBank No", or "Accession No". It is at this point that a decision has to be made: how to group the columns into data types. Some of the columns will become attributes as part of a complex data type grouping. Some will essentially become a class on their own. Figure 3 shows that "molecular_mass" and "amino_acid" have been made attributes of a complex element

Example3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<PP2-LikeProteins xmlns:xsdi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="PP2-LikeProteins1.xsd">
        <gene_description>
                <gene>At4g19840</gene>
                   <gene_properties>
                     <molecular_mass>28.1</molecular_mass>
                     <amino_acid>247</amino_acid>
                   </gene_properties>
            <source genBankNo="ncbi">
              <accessionNo>AV557741</accessionNo>
                </source>
                <species_name>
                        <species>Arabidopsis</species>
                          <family_name>
                                <family>Brassicaceae</family>
                           </family_name>
                </species_name>
        </gene_description>
</PP2-LikeProteins>
```

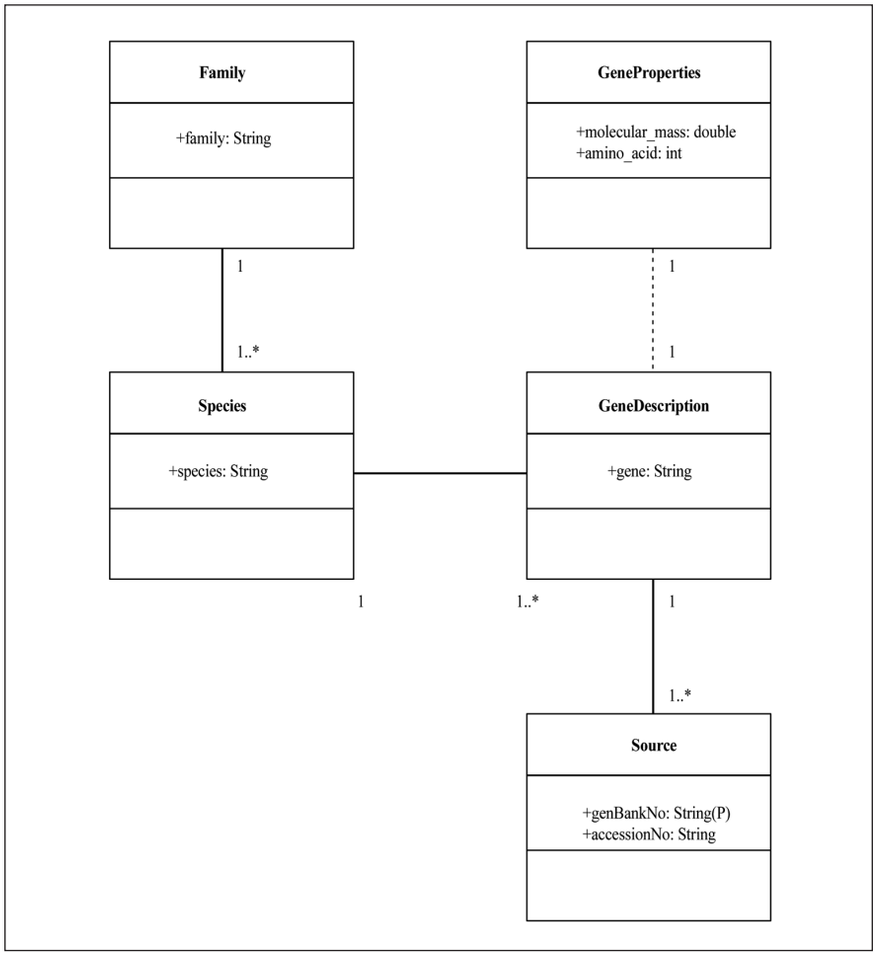

Figure 3. Conceptual-level UML diagram.

called "GeneProperties". On the other hand, "species" becomes an attribute of class "Species". In fact, column labels primarily identify attributes; we regard the values tabulated in the columns as instance values. Because we are dealing with two databases, we will denote the database ID by "GenBankNo" (which will have one of two values: "ncbi" or "tair") and the gene ID by "accessionNo". Both of these have data type String. Once we have identified our classes, we need to create a class diagram. Two classes are said to be associated if there is a relationship between their instances. Objects in Figure 3 have a one-to-many relationship denoted by (1..*), where "*" means unbounded. Another relationship between objects in Figure 3 is a one-to-one relationship. For example, the "GeneProperties" object is associated with one and only one "GeneDescription" object.

The XML Schema, Example3.xsd, corresponds directly to the objects and attributes in the conceptual-level diagram of Figure 3. We have used the XML Schema Definition Language, XSD (www.w3.org). The root element definition is the PP2-LikeProteins from Tables 1, 2, and 3. Next, we have the complex-type definitions, which are elements that contain child elements, attributes, or both. "GeneDescriptionType" is a complex data type that has elements "gene", "gene_properties", and "species". Element "GeneType" is a simple data type with the restriction that it has to be a String.

The XML document shown, Example3.xml, is only one of the many possible instance documents of the schema of Example3.xsd. This instance document is based on the data of Tables 1, 2, and 3. An instance document that conforms to the rules specified in the schema is said to be "valid". Note that it is possible to write an XML document with no schema. There are several very basic rules laid down by the W3C specification for XML (www.w3.org), (8). XML, which conforms to these basic rules, is said to be "well formed".

DISCUSSION

Designing an XML Schema can be quite a complex endeavor. The use of a

graphical environment, like UML, can aid in this process. In this paper, we have indicated how a bioinformatics XML Schema can be constructed with the aid of UML. The graphical nature of UML, and the fact that only a restricted subset of it is used, should make it possible for a domain expert such as a biologist to participate in the construction of the information model. XML Schemas allow the incorporation of object-oriented features into the design of the XML application. The information model chosen, of which the XML Schema is a big part, affects the type of queries possible and even influences database performance (9). Incorporating object-oriented principles makes the information model, and the database, easier to update.

REFERENCES

1. **Guerrini, V.H. and D. Jackson.** 2000. Bioinformatics and extended markup language (XML). J. Bioinformatics *1*:12-21.
2. **Booch, G., J. Rumbaugh, and I. Jacobson.** 1999. The Unified Modeling Language User Guide, 1st ed. Addison-Wesley, Boston, MA.
3. **Fowler, M. and K. Scott.** 2000. UML Distilled, 2nd ed. Addison-Wesley, Boston, MA.
4. **Routledge, N., L. Bird, and A. Goodchild.** 2002. UML and XML Schema: Thirteenth Australasian Database Conference (ADC), Melbourne, Australia, p. 1-10. *In* Xiafang Zhou (Ed.), Conferences in Research and Practice in Information Technology, vol. 5.
5. **Achard, F., G. Vaysseix, and E. Barillot.** 2001. XML, bioinformatics, and data integration. J. Bioinformatics *17*:115-125.
6. **Michael Hucka Systems Biology Workbench Development Group**. SCHUCS: a UML-based approach for describing data representations intended for XML encoding. 11 Dec 2000. Control and Dynamical Systems, MC 1-16. California Institute of Technology, Pasadena, CA.
7. **Satzinger, J.W. and T.U. Orvik.** 2001. The Object Oriented Approach: Concepts, System Development, and Modeling with UML, 2nd ed. Course Technology, Boston, MA.
8. **Birbeck, M., J. Diamond, J. Duckett, and O.G. Gudmundsson.** 2001. Professional XML, 2nd ed. Wrox Press Ltd., Birmingham, UK.
9. **Rubin, D.L., F. Shafa, D.E. Oliver, M. Hewett, and R.B. Altman.** 2002. Representing genetic sequence data for pharmacogenomics: an evolutionary approach using ontological and relational models. Bioinformatics *18(Suppl. 1)*:S207-S215.

Address correspondence to Dr. Russel Elton Bruhn, Associate Professor, Donaghey College of Information Science and Systems Engineering, University of Arkansas at Little Rock, Department of Information Science, 2801 South University Avenue, Little Rock, AR 72204, USA. e-mail: rebruhn@ualr.edu

Received 11 February 2003; accepted 3 April 2003.

Russel Elton Bruhn and Philip John Burton
University of Arkansas at Little Rock
Little Rock, AR, USA