1 *Hierarchies and Relationships*

1.1 Traditional Record Structures

One of the most common ways to represent information with computers is to use "records." Records are stored in a file, with one record per line. Such a file is called a *flat file*. A record consists of a series of data items, called "fields" or "columns." Here are some records from a health study (NHS 2004):

```
011500   18.66  0  0 62   46.271020111   25.220010
011500   26.93  0  1 63   68.951521001   32.651010
020100   33.95  1  0 65   92.532041101   18.930110
020100   17.38  0  0 67   50.351111100   42.160001
```

The actual records are considerably longer. It should be apparent that one cannot have any understanding of the meaning of the records without some explanation such as the following:

```
NAME      LENGTH  RANGE        FORMAT  MEAN OR CODES
instudy   6                    MMDDYY
bmi       8       13.25-60.07  Num     26.03
obesity   3       0-1          Num     0=No 1=Yes
ovrwt     8       0-1          Num     0=No 1=Yes
Height    3       49-79        Num     64.62
Wtkgs     8       38.1-175.1   Num     70.2
Weight    3       84-386       Num     154.75


NAME      LABEL
instudy   Date of randomization into study
bmi       Body Mass Index.  Weight(kgs)/height(m)**2
```

```
obesity  Obesity (30.0 <= BMI)
ovrwt    Overweight (25 <= BMI < 30)
Height   Height (inches)
Wtkgs    Weight (kilograms)
Weight   Weight (pounds)
```

The explanation of what the fields mean is called *metadata*. In general, metadata are any "data about data," such as the names of the fields, the kind of values that are allowed, the range of values, and explanations of what the fields mean.

In this case each field has a fixed number of characters, and each record has a fixed total number of characters. This is called the *fixed-width format* or *fixed-column format*. This format simplifies the processing of the file, but it limits what can be said within each field. If the text that should be in a field does not fit, then it must be abbreviated or truncated. There are other file formats that eliminate these limitations. One commonly used format is to use commas or tabs to delimit the fields. This allows the fields to have varying size. However, it complicates processing when the delimiting character (i.e., the comma or tab) must be used within a field.

The information in the record is often highly redundant. For example, the *obesity* and *ovrwt* fields are unnecessary because they can be computed from the *bmi* field. Similarly, the *bmi* field can be computed from the *Height* and *Weight* fields. Another common feature of flat files is that the field formats are often inappropriate. For example, the *obesity* field can only have the values "yes" or "no," but it is represented using numbers.

Each field of a flat file is defined by features such as its name, format, description, and so on. A *database* is a collection of flat files (called *tables*) with auxiliary structures (e.g., indexes) that improve performance for certain commonly used operations. The description of the fields of one or more flat files is called the *schema*.

A database schema is an example of an *ontology*. In general, whenever data are structured, the description of their structure is the ontology for the data. A glance at the example record makes it clear that the raw data record is completely useless without the ontology. The ontology is what gives the raw data their meaning. The same is true for any kind of data, whether they be electronic data used by a computer or audiovisual data sensed by a person. Ontologies are the means by which a person or some other agent understands its world, as well as the means by which a person or agent communicates with others.

Summary

- A flat file is a collection of records.
- A record consists of fields.
- Each record in a flat file has the same number and kinds of fields as any other record in the same file.
- The schema of a flat file describes the structure (i.e., the kinds of fields) of each record.
- A schema is an example of an ontology.

1.2 The eXtensible Markup Language

Flat files are simple and easy to process. A typical program using and producing flat files simply performs the same operation on each record. However, flat files are limited to relatively simple forms of data. They are not well suited to the complex information of genomics, proteomics, and so on. Accordingly, a new approach is necessary.

The eXtensible Markup Language (XML) is a powerful and flexible mechanism that can be used to represent bioinformatic data and facilitates communication. Unlike flat files, an XML document is *self-describing*: the name of each attribute is specified in addition to the value of the attribute. The health study record shown above could be written like this in XML:

```
<Interview RandomizationDate="2000-01-15" BMI="18.66" Height="62".../>
<Interview RandomizationDate="2000-01-15" BMI="26.93" Height="63".../>
<Interview RandomizationDate="2000-02-01" BMI="33.95" Height="65".../>
<Interview RandomizationDate="2000-02-01" BMI="17.38" Height="67".../>
```

The basic unit of an XML document is called an *element*. It is analogous to a record in a flat file, except that a single XML document can have many kinds of element. One would need a large collection of flat files (or a database with many tables) to represent the elements of a *single* XML document, and even that would not capture all of it, because the kinds of element in an XML document can be intermixed. Each kind of element is labeled by a name called its *tag*. The example given above is an `Interview` element.

The fields of an XML element are called its *attributes*. Flat files generally distinguish fields from one another by their positions in the record. XML attributes can appear in any order, and an attribute that is not needed by an element is not written at all.

An attribute in general is a property or characteristic of an entity. Linguistically, attributes are adjectives that describe entities. For example, a person may be overweight or obese, and the `BMI` attribute makes the description quantitative rather than qualitative. The notion of attribute represents two somewhat different concepts: the attribute in general and the attribute of a specific entity. `BMI` is an example of an attribute, but one would also speak of a `BMI` equal to 18.66 for a specific person as being an attribute. To avoid confusion we will refer to the former as the *attribute name*, while the latter is an *attribute value*.

```
<!ATTLIST molecule
             title      CDATA   #IMPLIED
             id         CDATA   #IMPLIED
             convention CDATA   "CML"
             dictRef    CDATA   #IMPLIED
             count      CDATA   #REQUIRED
>
```

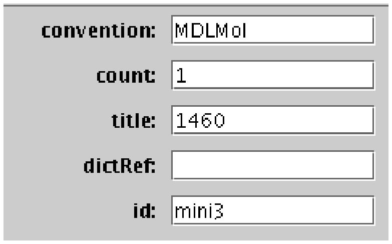
Figure 1.1 Part of the Chemical Markup Language DTD. This defines the attribute names that are allowed in a `molecule` element.

Just as a database is described by its schema, an XML document is described by its Document Type Definition (DTD). The DTD specifies the attribute names that are allowed for each kind of element. For example, in the Chemical Markup Language (CML) (CML 2003), a `molecule` can have a title, identifier, convention, dictionary reference, and count. Figure 1.1 shows how this is specified in the CML DTD. A `#REQUIRED` attribute is one that must be specified in every element of this kind; an `#IMPLIED` attribute is optional. If a value is specified in the DTD, then it is the default value of the attribute. For example, if no `convention` is specified, then it has the value "`CML`." CDATA means "character data" which means that the attribute value can use any kind of text except for elements.

One enters or updates data for an XML element in the same manner that one enters or updates data for a database table. An example of such a data entry screen is given in figure 1.2.

XML reserves two characters for indicating the presence of markup. The left angle bracket (<) is used by XML to mark the beginning of each element.

It is also used to show where an element ends. To include the left angle bracket in ordinary text, write it as "`&lt;`". Writing a special character like the left angle bracket as "`&lt;`" is called *escaping*. The ampersand character (`&`) is also reserved by XML, and it must be written as "`&amp;`".



convention:	MDLMol
count:	1
title:	1460
dictRef:	
id:	mini3

Figure 1.2 Data entry screen for the `molecule` element of the Chemical Markup Language.

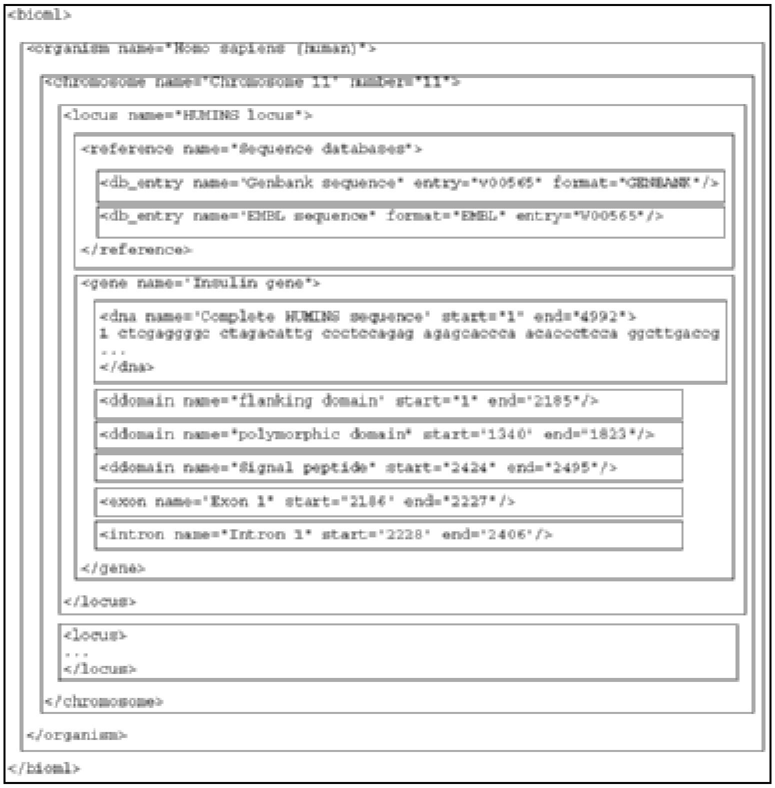
Summary

- XML is a format for representing data.
- An XML element is analogous to a record in a flat file.
- An XML attribute is analogous to a field of a record.
- An XML DTD is a schema that describes the structure of the elements of an XML file.

1.3 Hierarchical Organization

Modern biology and medicine, like much of society, is currently faced with overwhelming amounts of raw data being produced by new information-gathering techniques. In a relatively short period of time information has gone from being relatively scarce and expensive to being plentiful and inexpensive. As a consequence, the traditional methods for dealing with information are overwhelmed by the sheer volume of information available. The traditional methods were developed when information was scarce, and they cannot handle the enormous scale of information.

The first and most natural reaction by people to this situation is to attempt to categorize and classify. People are especially good at this task. We are constantly categorizing objects, experiences, and people. We do it effortlessly and unconsciously. The very words we use to express ourselves represent categories. It is only when a categorization is problematic that we notice that we have been categorizing at all. Biology was the first discipline to engage in systematic, large-scale classification because of the enormous complexity of its domain.



```
<bioml>
  <organism name="Homo sapiens (human)">
    <chromosome name="Chromosome 11" number="11">
      <locus name="HUMINS locus">
        <reference name="Sequence databases">
          <db_entry name='Genbank sequence" entry="V00565" format="GENBANK"/>
          <db_entry name='EMBL sequence" format="EMBL" entry="V00565"/>
        </reference>
        <gene name='Insulin gene">
          <dna name='Complete HUMINS sequence' start="1" end="4992">
          1 ctcgaggggc ctagacattg ccctccagag agagcaccca acaccctcca ggcttgaccg
          ...
          </dna>
          <ddomain name="flanking domain' start="1" end='2185"/>
          <ddomain name="polymorphic domain" start='1340' end="1823"/>
          <ddomain name="Signal peptide" start="2424" end="2495"/>
          <exon name='Exon 1" start="2186' end="2227"/>
          <intron name="Intron 1" start='2228' end='2406'/>
        </gene>
      </locus>
      <locus>
      ...
      </locus>
    </chromosome>
  </organism>
</bioml>
```

Figure 1.3 A BioML document showing some of the information about the human insulin gene. Boxes were drawn around each XML element so that the hierarchical structure is more apparent. XML documents normally indicate the hierarchical structure by successive indentation, as in this example.

What makes XML powerful is the ability to organize data hierarchically. XML elements are much more than just self-describing records; each element can contain other elements, which can contain other elements, to arbitrary depth. Figure 1.3 shows a small part of the genomic data for the insulin gene represented using the Biopolymer Markup Language (BioML) (BioML 2003). This XML document consists of a `bioml` element containing an `organism` element. The `organism` element, in turn, contains `chromosome` elements, which contain `locus` elements, which contain `genes`, which contain the DNA sequence, domains, exons, introns, and so on. Along the way, the elements also contain references to database entries that furnish the source material for the genomic information. This example shows the organization of information about biopolymers starting at the organism level and successively elaborating until one sees individual DNA bases.

Because of the hierarchical nature of an XML document, there is always a "top" of the hierarchy called the *root*. In figure 1.3 the root is the `bioml` element. The root is split into a series of branches, which in turn split into branches, and so on, like the branching of a tree. The terminology of family trees is commonly used for the relationships within the hierarchy. The elements contained in an element are called its *child* elements, and the containing element is the *parent*. The children of the same parent are *siblings*. Note that this family tree is asexual: each element (except for the root) has exactly one parent.

The tags and attributes occurring in an XML DTD constitute the *vocabulary* of the ontology. When one is creating an ontology it is important to choose the tags and attributes so that they correspond to how people use the words. Ontologies should facilitate communication between people as well as between computer systems. Because of this emphasis on communication, ontologies are often referred to as *languages*. Ontologies are also important for information retrieval from databases, and the terminology in an ontology is called a *controlled vocabulary* in this context. An ontology is a specialized language for communication in a particular domain. The communication can be between people, between people and computers, or between computers. Ontologies based on XML are more specifically called *markup languages* because of the historical origin of XML as a means of marking up text for the purpose of typesetting documents. Thus the "ML" in BioML and CML both stand for "Markup Language" even though neither of these ontologies is concerned with typesetting.

Summary

- Classification is one way in which people organize a domain in order to understand it more easily.

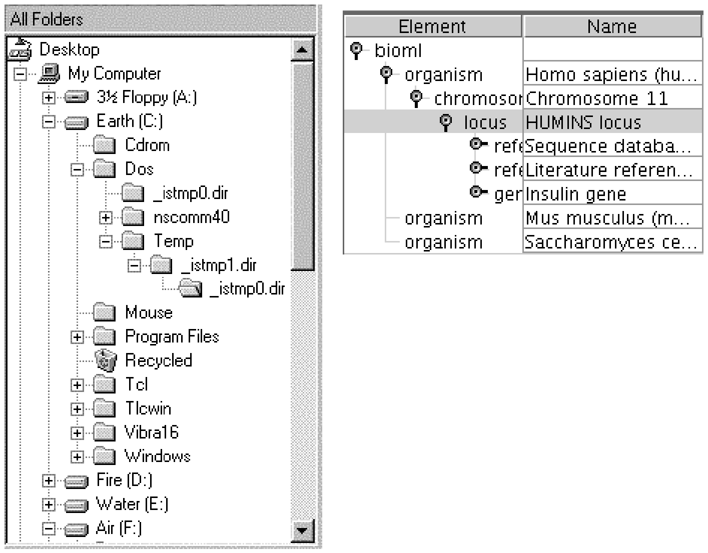
- Classifications are frequently organized in the form of a hierarchy.
- XML elements are hierarchical: each element can contain other elements, that in turn can contain other elements, and so on.

1.4 Creating and Updating XML

This little example illustrates how statements can be much worse than just being false: they can be meaningless. One of the main functions of a good ontology is that it limits what can be said, so that statements using the ontology always make sense to a member of the community served by the ontology. This is done by using *constraints*. Some constraints have already been discussed in section 1.2 where we saw that one can specify what attributes are allowed for each kind of element. One can also specify which elements can be contained in other elements as well as how many are allowed. These constraints are especially useful when one is creating and updating XML documents, and that is the topic of this section.

Viewing and updating an XML document may seem to be a formidable task, but one rarely looks directly at an XML document any more than one would look at the page source of an HTML (Hypertext Markup Language) document. One uses an XML tool for creating, viewing, and updating. The single term "editing" is used for all three of these activities. An XML editor is a tool that supports the editing of an XML document. XML editors automatically take care of routine tasks such as escaping special characters and making sure that the document is consistent. There are many such tools available. The examples in this book used Xerlin (Xerlin 2003), an open source XML editor that is available from the Apache project. XML viewers and editors make good use of the hierarchical structure of an XML document. This structure is analogous to a file folder or directory structure: The XML document is viewed and updated in much the same way as files in a directory. In figure 1.4 one can see a typical file manager compared with an XML document editor showing the BioML insulin gene document.

The DTD of an XML document specifies more than just the attributes of each element. For example, in CML, a `molecule` contains an `atomArray` and a `bondArray`, and they must occur in this order: the `atomArray` must occur first, and the comma indicates that the `bondArray` must occur second. An `atomArray` element contains one or more `atom` elements, and a `bondArray` consists of one or more `bond` elements. The DTD would specify this as follows:

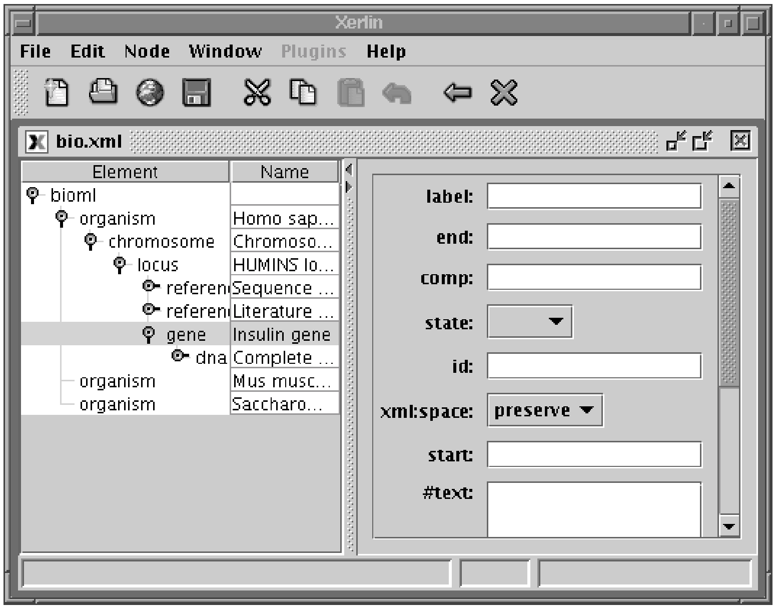


Element	Name
bioml	
organism	Homo sapiens (hu...
chromosor	Chromosome 11
locus	HUMINS locus
refe	Sequence databa...
refe	Literature referen...
ger	Insulin gene
organism	Mus musculus (m...
organism	Saccharomyces ce...

Figure 1.4 File management vs. XML document management. The image on the left used the Windows file manager. It shows disk drives, folders, and files on a PC. The image on the right used the Xerlin XML document editor. It shows the elements of a single XML document.

```
<!ELEMENT molecule (atomArray,bondArray)>
<!ELEMENT atomArray (atom+)>
<!ELEMENT bondArray (bond+)>
```

The ELEMENT statements above determine the *content* of these elements. A specification such as (atom+) is called a content model. The ATTLIST statement for molecule given earlier determines the attributes that can be in an element. A DTD will normally have one ELEMENT statement and one ATTLIST statement for each kind of element that can be in the document. A more complete DTD for molecules is shown in figure 1.6. Because the same attributes are allowed in many elements, a DTD can be very long. ENTITY statements are a method for simplifying the writing of DTDs, by allowing one to specify content and lists of attributes just once. In figure 1.7 two entities were defined and then used several times. Large DTDs such as CML use a large number of entities. These are just two of the entities in CML.



Element	Name
bioml	
organism	Homo sap…
chromosome	Chromoso…
locus	HUMINS lo…
referen	Sequence …
referen	Literature …
gene	Insulin gene
dna	Complete …
organism	Mus musc…
organism	Saccharo…

xml:space: preserve

Figure 1.5 Data entry screen for an element of an XML document. The window on the left shows the hierarchical structure of the XML document in the same manner as a file manager. A `gene` element is highlighted, indicating that this is the currently open element. The attributes for the `gene` element are shown in the right window. The window on the right acts like a data entry screen for viewing and updating the attributes of the element.

```
<!ELEMENT molecule (atomArray, bondArray)>
<!ATTLIST molecule
        title           CDATA   #IMPLIED
        id              CDATA   #IMPLIED
        convention      CDATA   "CML"
        dictRef         CDATA   #IMPLIED
        count           CDATA   "1"
>
<!ELEMENT atomArray (atom+)>
<!ATTLIST atomArray
        title           CDATA   #IMPLIED
        id              CDATA   #IMPLIED
        convention      CDATA   "CML"
```

```
>
<!ELEMENT atom EMPTY>
<!ATTLIST atom
        elementType     CDATA   #IMPLIED
        title           CDATA   #IMPLIED
        id              CDATA   #IMPLIED
        convention      CDATA   "CML"
        dictRef         CDATA   #IMPLIED
        count           CDATA   "1"
>
<!ELEMENT bondArray (bond+)>
<!ATTLIST bondArray
        title           CDATA   #IMPLIED
        id              CDATA   #IMPLIED
        convention      CDATA   "CML"
>
<!ELEMENT bond EMPTY>
<!ATTLIST bond
        title           CDATA   #IMPLIED
        id              CDATA   #IMPLIED
        convention      CDATA   "CML"
        dictRef         CDATA   #IMPLIED
        atomRefs        CDATA   #IMPLIED
>
```

Figure 1.6 Part of the Chemical Markup Language DTD. This part defines the content and some of the attributes of a `molecule` element as well as the content and some of the attributes of elements that can be contained in a `molecule` element.

```
<!ENTITY % title_id_conv '
        title           CDATA   #IMPLIED
        id              CDATA   #IMPLIED
        convention      CDATA   "CML"    '>
<!ENTITY % title_id_conv_dict
%title_id_conv;
```

```
        'dictRef         CDATA   #IMPLIED'>

<!ELEMENT molecule (atomArray, bondArray)>
<!ATTLIST molecule
%title_id_conv_dict;
        count           CDATA   "1"
>
<!ELEMENT atomArray (atom+)>
<!ATTLIST atomArray
%title_id_conv;
>
<!ELEMENT atom EMPTY>
<!ATTLIST atom
        elementType     CDATA   #IMPLIED
%title_id_conv_dict;
        count           CDATA   "1"
>
<!ELEMENT bondArray (bond+)>
<!ATTLIST bondArray
%title_id_conv;
>
<!ELEMENT bond EMPTY>
<!ATTLIST bond
%title_id_conv_dict;
        dictRef         CDATA   #IMPLIED
        atomRefs        CDATA   #IMPLIED
>
```

Figure 1.7 Part of the Chemical Markup Language DTD. This DTD uses entities to simplify the DTD in figure 1.6.

In addition to simplifying a DTD, there are other uses of XML entities:

- Entities can be used to build a large DTD from smaller files. The entities in this case refer to the files being incorporated rather than to the actual value of the entity. Such an entity would be defined like this:

```
  <!ENTITY % dtd1 SYSTEM "ml.dtd">
```

To include the contents of the file `ml.dtd`, one uses `%dtd1;` in your DTD. One can use a URL instead of a filename, in which case the DTD information will be obtained from an external source.

- Entities can be used to build a large document from smaller documents. The smaller documents can be local files or files obtained from external sources using URLs. For example, suppose that an experiment is contained in five XML document files. One can merge these files into a single XML document as follows:

```
<?xml version="1.0"?>
<!DOCTYPE ExperimentSet SYSTEM "experiment.dtd"
[
  <!ENTITY experiment1 SYSTEM "experiment1.xml">
  <!ENTITY experiment2 SYSTEM "experiment2.xml">
  <!ENTITY experiment3 SYSTEM "experiment3.xml">
  <!ENTITY experiment4 SYSTEM "experiment4.xml">
  <!ENTITY experiment5 SYSTEM "experiment5.xml">
]>
<ExperimentSet>
  &experiment1;
  &experiment2;
  &experiment3;
  &experiment4;
  &experiment5;
</ExperimentSet>
```

  Note that entities used within documents use the ampersand rather than the percent sign. This example is considered again in section 11.6 where it is discussed in more detail.

When one is editing an XML document, the DTD assists one to identify the attributes and elements that need to be provided. Figure 1.5 shows the BioML insulin gene document. The "directory" structure is on the left, and the attributes are on the right. In this case a `gene` element is open, and so the attributes for the gene element are displayed. To enter or update an attribute, click on the appropriate attribute and use the keyboard to enter or modify the attribute's value. When an attribute has only a limited list of possible values, then one chooses the desired value from a menu. Attributes are specified in the same manner as one specifies fields of a record in a database using a traditional "data entry" screen. An XML document is effectively an entire database with one table for every kind of element.

In addition to attributes, an XML element can have text. This is often referred to as its *text content* to distinguish it from the elements it can contain. In an XML editor, the text content is shown as if it were another child element, but labeled with `#text`. It is also shown as if it were another attribute, also labeled with `#text`.

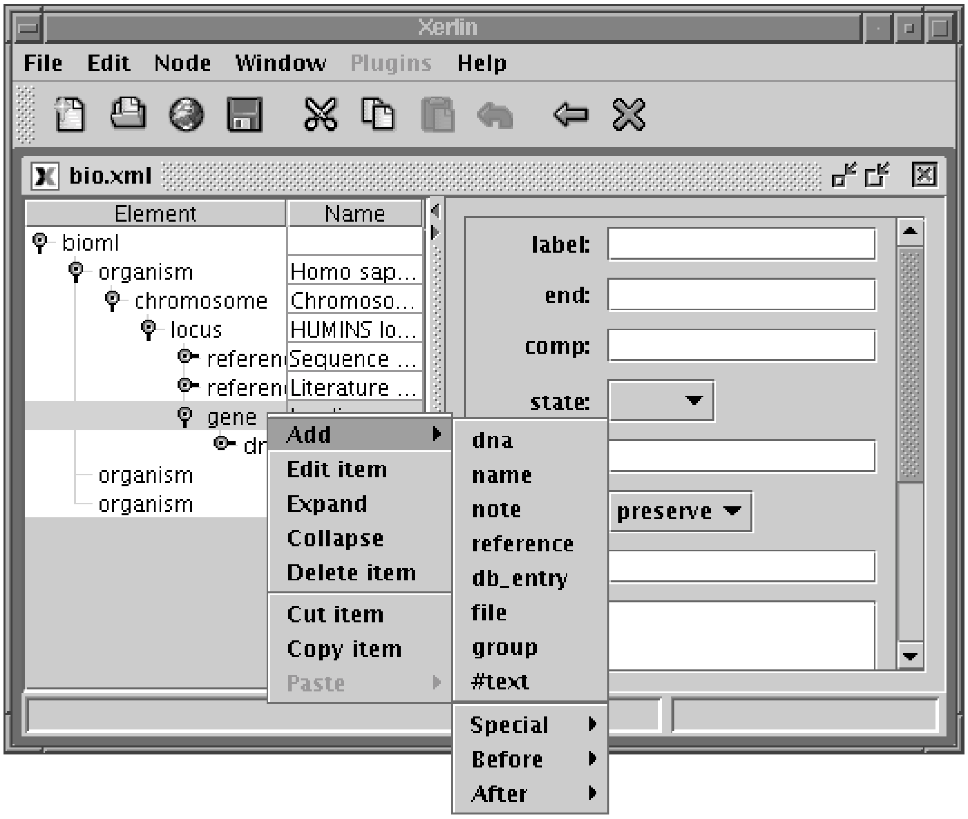


Figure 1.8 The process of adding a new element to an XML document. The menus shown were obtained by right-clicking on the `gene` element and then selecting the `Add` choice. The menu containing `dna`, `name`, and so on shows the elements that are allowed in the `gene` element.

The hierarchical structure of an XML document is manipulated by opening and closing each element just as one opens and closes directories (folders) when managing a collection of files. Unlike file management, the DTD of an XML document can be very precise about what elements are allowed to be in another element, as well as the order in which they must appear. Figure 1.8 shows how to add an element to the `gene` element of a BioML document. Click (with the right mouse button) on the `gene` element and a menu appears that allows one to `Add` another element. Selecting this choice in the menu displays another menu. This second menu shows all the kinds of child elements that are allowed at this time in the parent element. In figure 1.9 a `note` element was chosen. Figure 1.10 shows the result of making this selection: the `gene` element now has a new `note` child element. The right window now displays the attributes for the newly created element.

The elements that are allowed in an element depend on the editing context. The `molecule` element must have exactly two child elements: `atomArray` and `bondArray`. As a result, when a `molecule` is selected the first time, the only choice for a child element will be `atomArray`. After adding such an element, the only choice will be `bondArray`.

The file directory metaphor is a compelling one, but it is important to note how XML differs from a directory structure. The differences are explained in table 1.1. Just as a database schema is an example of an ontology, an XML DTD is also an example of an ontology. However, unlike a database, an XML document is self-describing. One can understand much of the meaning of the data without recourse to the ontology. Indeed, there are tools which can guess the DTD for an XML document that does not have one.

Summary

- XML documents are examined and updated by taking advantage of the hierarchical structure.

- The XML DTD assists in updating a document by giving clues about what attributes need to be entered as well as what elements need to be added.

1.5 The Meaning of a Hierarchy

Each kind of element in an XML document represents a *concept*. Concepts are the means by which people understand the world around them. They classify the world into units that allow comprehension. They also make it

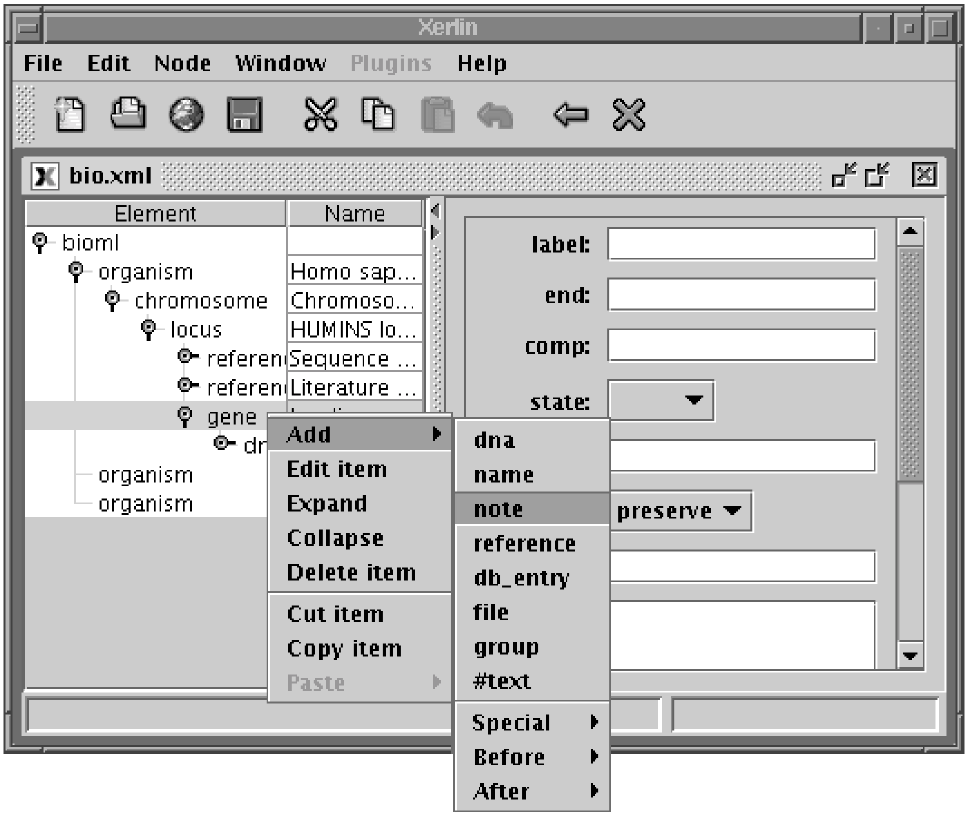
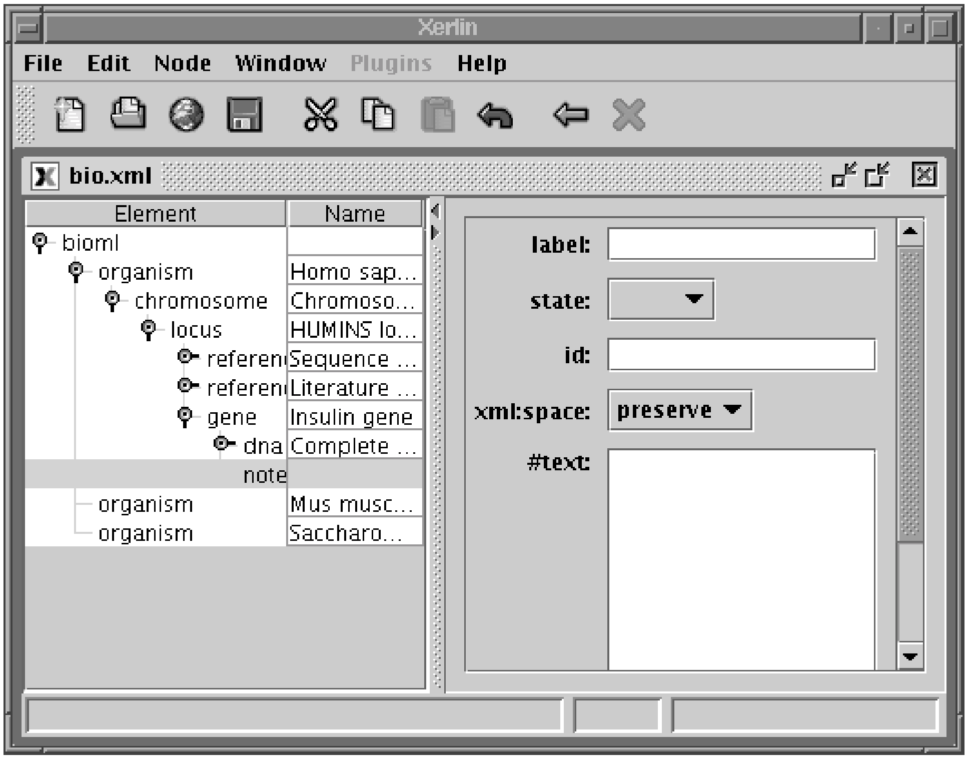


Figure 1.9 Adding a new element to an XML document. A `note` element has been chosen to be added to a `gene` element.

possible for people to communicate with each other. Individuals must have a shared conceptual framework in order to communicate, but communication requires more than just a shared conceptualization; it is also necessary for the concepts to have names, and these names must be known to the two individuals who are communicating.

Biochemistry has a rich set of concepts ranging from very generic notions such as *chemical* to exquisitely precise notions such as *Tumor necrosis factor alpha-induced protein 3*. Concepts are typically organized into hierarchies to capture at least some of the relationships between them. XML document hierarchies are a means by which one can represent such hierarchical organi-



Element	Name
bioml	
organism	Homo sap…
chromosome	Chromoso…
locus	HUMINS lo…
referen	Sequence …
referen	Literature …
gene	Insulin gene
dna	Complete …
note	
organism	Mus musc…
organism	Saccharo…

label:
state:
id:
xml:space: preserve
#text:

Figure 1.10 Result of adding a new element to an XML document. A `note` element has been added to a `gene` element. The active element is now the `note` element, and its attributes appear in the window on the right side.

zations of knowledge.

Aristotle (384-322 BC) was the first who understood the difficulty of categorizing living organisms into classes according to their anatomical and physiological characteristics (Asimov 1964). Since then, this tradition of classification has been one of the major themes in science. Figure 1.11 illustrates a hierarchy of chemicals taken from EcoCyc (EcoCyc 2003). For example, *protein* is more specific than *chemical*, and *enzyme* is more specific than *protein*. Classifications that organize concepts according to whether concepts are more general or more specific are called *taxonomies* by analogy with biological classifications into species, genera, families, and so on.

Hierarchies are traditionally obtained by starting with a single all-inclu-

File Manager	XML Editor
A file or directory (folder) name uniquely identifies it. Such a name is not well suited for describing the contents, or for specifying what it means.	An XML element tag is for specifying what the element means, not how to obtain it.
In a directory each file or other directory within it must have a unique name.	In an XML element one can have more than one child element with the same tag.
There are essentially no constraints on what names can be used, as long as they are unique within the directory.	XML elements can only have tags that are allowed by the DTD.
The attributes of files and directories are always the same, and serve only for administrative purposes by the operating system.	XML elements can have any attributes that are allowed by the DTD.
File names are sometimes case insensitive. Case insensitivity means that there is no difference between upper- and lower-case letters.	XML is case sensitive. Upper- and lower-case letters are different.

Table 1.1 Comparison of directory/file management with XML document editing

sive class, such as "living being," and then subdividing into more specific subclasses based on one or more common characteristics shared by the members of a subclass. These subclasses are, in turn, subdivided into still more specialized classes, and so on, until the most specific subclasses are identified. We use this technique when we use an outline to organize a task: the most general topic appears first, at the top of the hierarchy, with the more specialized topics below it. Constructing a hierarchy by subdivision is often called a "top-down" classification.

An alternative to the top-down technique is to start with the most specific classes. Collections of the classes that have features in common are grouped together to form larger, more general, classes. This is continued until one collects all of the classes together into a single, most general, class. This approach is called "bottom-up" classification. This is the approach that has been used in the classification of genes (see figure 1.12). Whether one uses a

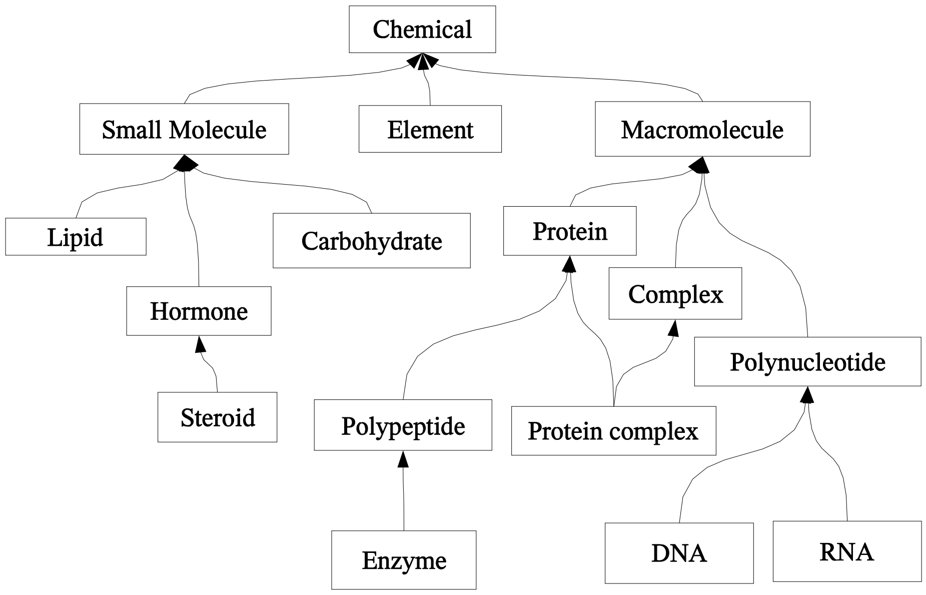


Figure 1.11 Chemical hierarchy (EcoCyc 2003).

top-down or bottom-up technique, it is always presumed that one can define every class using shared common characteristics of the members.

There are many algorithms for constructing hierarchical classifications, especially taxonomies, based on attributes of entities (Jain and Dubes 1988). Such algorithms are usually referred to as *data-clustering* algorithms. An example of a hierarchy constructed by a data-clustering algorithm is shown in figure 5.3. The entities being clustered in this case are a set of genes, and the hierarchy appears on the left side of the figure. Such automated classifications have become so routine and common that many tools construct them by default.

The notions of taxonomy and hierarchy have been an accepted part of Western civilization since the time of Aristotle. They have been a part of this culture for so long that they have the status of being completely obvious and natural. Aristotle already emphasized that classifications must be "correct," as if they had the status of a law of nature rather than being a means for understanding the world. This attitude toward classification was not questioned until relatively recently, and is still commonly accepted. By the middle of the nineteenth century, scholars began to question the implicit assumptions underlying taxonomic classification. Whewell, for example, discussed classification in science, and observed that categories are not usually specifiable by shared characteristics, but rather by resemblance to what he called "paradigms." (Whewell 1847) This theory of categorization is now called "prototype theory." A *prototype* is an ideal representative of a category from which other members of the category may be derived by some form of modification. One can see this idea in the classification of genes, since they evolve via mutation, duplication, and translocation (see figure 1.13). Wittgenstein further elaborated on this idea, pointing out that various items included in a category may not have one set of characteristics shared by all, yet given any two items in the category one can easily see their common characteristics and understand why they belong to the same category (Wittgenstein 1953). Wittgenstein referred to such common characteristics as "family resemblances," because in a family any two members will have some resemblance, such as the nose or the eyes, so that it is easy to see that they are related, but there may be no one feature that is shared by all members of the family. Such a categorization is neither top-down nor bottom-up, but rather starts somewhere in the middle and goes up and down from there.

This is especially evident in modern genetics. Genes are classified both by function and by sequence. The two approaches interact with one another in complex ways, and the classification is continually changing as more is learned about gene function. Figure 1.12 shows some examples of the classification of genes into families and superfamilies. The superfamily is used to describe a group of gene families whose members have a common evolutionary origin but differ with respect to other features between families. A gene family is a group of related genes encoding proteins differing at fewer than half their amino acid positions. Within each family there is a structure that indicates how closely related the genes are to one another. For example figure 1.13 shows the evolutionary structure of the nuclear receptor gene family. The relationships among the various concepts is complex, including evolution, duplication and translocation.

The hierarchies shown in figure 1.11, 1.12, and 1.13 are very different from one another due to the variety of purposes represented in each case. The chemical hierarchy in figure 1.11 is a specialization/generalization hierarchy. The relationship here is called *subclass* because mathematically it represents a subset relationship between the two concepts. The gene families and superfamilies in figure 1.12 are also related by the subclass relationship, but the

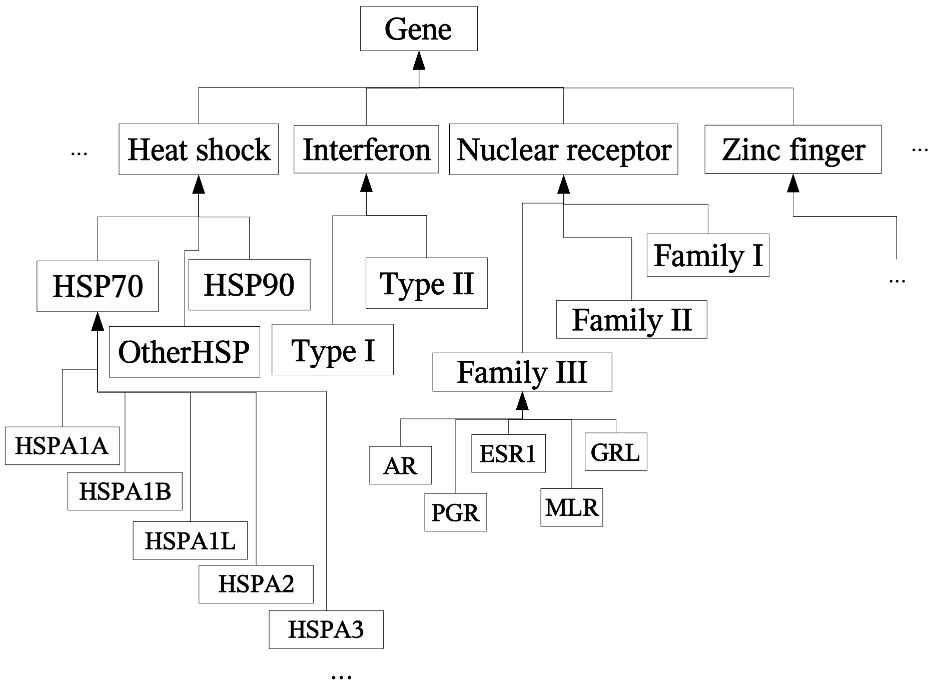


Figure 1.12 Some gene families. The first row below `Gene` in this classification consists of superfamilies. The row below that contains families. Below the families are some individual genes. See (Cooper 1999), Chapter 4.

individual genes shown in the diagram are members (also called *instances*) of their respective families rather than being subsets. However, the nuclear receptor gene diagram in figure 1.13 illustrates that the distinction between subclass and instance is not very clear-cut, as the entire superfamily evolved from a single ancestral gene. In any case, the relationships in this last diagram are neither subclass nor instance relationships but rather more complex relationships such as: *evolves by mutation*, *duplicates*, and *translocates*.

Although hierarchical classification is an important method for organizing complex information, it is not the only one in common use. Two other techniques are partitioning and self-organizing maps. Both of these can be regarded as classification using attribute values rather than hierarchical structures. In *partitioning*, a set of entities is split into a specified number of subsets (MacQueen 1967). A *self-organizing map* is mainly used when a large num-

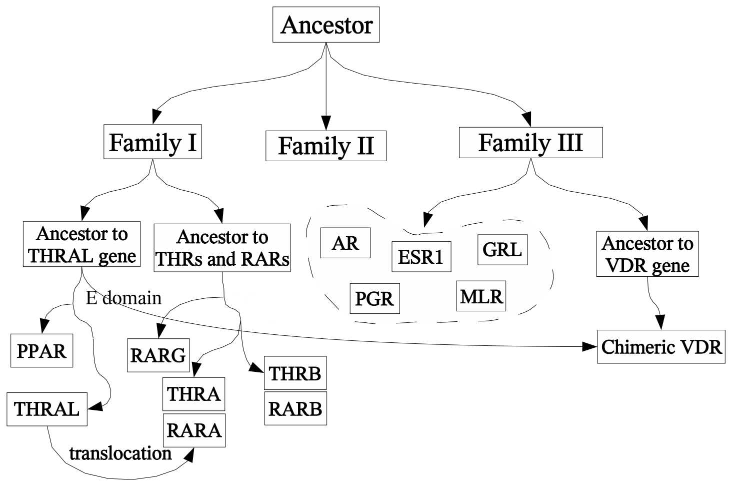


Figure 1.13 The human nuclear receptor gene superfamily. A common ancestor evolved into the three gene families. Unlabeled arrows represent evolution over time. Labeled arrows indicate translocation between families or subfamilies. See (Cooper 1999), Figure 4.28.

ber of attributes for a set of entities is to be reduced to a small number of attributes, usually two or three (Kohonen 1997). The attributes are then displayed using visual techniques that make the clusters easy for a person to see. There are also many clustering techniques that combine some or all of these techniques.

Summary

- Classifications can be constructed top-down, bottom-up, or from the middle.

- Classifications can be based on many principles: subclass (subset), instance (member), or more complex relationships.

- It is even possible for a classification to be based on several relationships at the same time.

1.6 Relationships

Of course, most titles are like this, and the abstract quickly clears up the confusion. However, it does point out how important such connecting phrases can be to the meaning of a document. These are called *relationships*, and they are the subject of this section.

The organization of concepts into hierarchies can capture at least some of the relationships between them, and such a hierarchy can be represented using an XML document hierarchy. The relationship in an XML document between a parent element and one of its child elements is called *containment* because elements contain each other in the document. However, the actual relationship between the parent element and child element need not be a containment. For example, it is reasonable to regard a `chromosome` as containing a set of `locus` elements because a real chromosome actually does contain loci. Similarly, a gene really does contain exons, introns, and domains. However, the relationship between a `gene` and a `reference` is not one of containment, but rather the *referral* or *citation* relationship.

One of the disadvantages of XML is that containment is the only way to relate one element to another explicitly. The problem is that all the various kinds of hierarchy and various forms of relationship have to be represented using containment. The hierarchy in figure 1.13 does not use any relationships that could reasonably be regarded as being containment. Yet, one must use the containment relationship to represent this hierarchy. The actual relationship is therefore necessarily implicit, and some auxiliary, informal technique must be used to elucidate which relationship is intended.

Unfortunately, this is not a small problem. One could not communicate very much if all one had were concepts and a single kind of relationship. Relating concepts to each other is fundamental. Linguistically, concepts are usually represented by nouns and relationships by verbs. Because relationships relate concepts to concepts, the linguistic notion of a simple sentence, with its subject, predicate, and object, represents a basic fact. The subject and object are the concepts and the predicate is the relationship that links them.

One can specify relationships in XML, but there are two rather different ways that this can be done, and neither one is completely satisfactory. The first technique is to add another "layer" between elements that specifies the relationship. This is called *striping*. A BioML document could be represented using striping, as in figure 1.14. If one consistently inserts a relationship element between parent and child concept elements, then one can unambiguously distinguish the concept elements from the relationship elements.

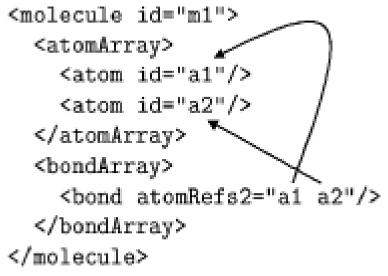
Striping was first introduced in the Resource Description Framework (RDF) (Lassila and Swick 1999).

```
...
   <locus name="HUMINS locus">
     <contains>
       <gene name="Insulin gene">
         <isStoredIn>
           <db_entry name="Genbank sequence" entry="v00565"
                     format="GENBANK"/>
           <db_entry name="EMBL sequence" format="EMBL"
                     entry="V00565"/>
         </isStoredIn>
         <isCitedBy>
           <db_entry name="Insulin gene sequence" format="MEDLINE"
                     entry="80120725"/>
           <db_entry name="Insulin mRNA sequence" format="MEDLINE"
                     entry="80236313"/>
           <db_entry name="Localization to Chromosome 11" format="MEDLINE"
                     entry="93364428"/>
         </isCitedBy>
         <hasSequence>
           <dna name="Complete HUMINS sequence" start="1" end="4992">
             1 ctcgaggggc ctagacattg ccctccagag agagcaccca acaccctcca ggcttgaccg
             ...
           </dna>
         </hasSequence>
       </gene>
     </contains>
   </locus>
...
```

Figure 1.14 Using striping to represent relationships involving the human insulin gene. The shaded elements in the figure are the relationships that link a parent element to its child elements.

Another way to specify a relationship is to use a *reference*. A reference is an attribute of an XML element that refers to some other data. The referenced data can be anything and anywhere, not just XML elements and not just in the same XML document. This technique is much more flexible and powerful than striping. An example of a molecule with two atoms bound to each other is shown in figure 1.15. The two atoms in the `atomArray` are referenced by the `bond` in the `bondArray`. In general, a reference could be to anything that has a URI.

Striping and references can be used in the same document. In RDF, the two techniques can be used interchangeably, and they have exactly the same



```
<molecule id="m1">
  <atomArray>
    <atom id="a1"/>
    <atom id="a2"/>
  </atomArray>
  <bondArray>
    <bond atomRefs2="a1 a2"/>
  </bondArray>
</molecule>
```

Figure 1.15 The use of references to specify a bond between two atoms in a molecule. The arrows show the atoms that are being referenced by the bond element.

meaning. A relationship specified with either striping or a reference forms a statement. For example, in figure 1.14 there is the statement "The human insulin gene is cited by db entry 80129725." Both striping and references help organize XML documents so that relationships are explicit. They contribute to the goal of ensuring that data are self-describing. References are commonly used in bioinformatics ontologies, but striping is seldom used outside of RDF ontologies.

One feature of RDF that makes it especially attractive is that its semantics have been formalized using mathematical logic. There are now a number of ontology languages that extend RDF and that also have formal semantics. The DARPA Agent Markup Language (DAML) is a DARPA project that produced the DAML+OIL language. This language has recently been superseded by the Web Ontology Language (OWL). OWL is a standard of the World Wide Web Consortium (W3C). The RDF and OWL standards are available on the W3C website (`www.w3c.org`). Both RDF and OWL will be discussed in much more detail in the rest of this book.

Summary

- Relationships connect concepts to each other.
- XML has only one explicit kind of relationship: containment.
- Relationships can be specified in XML in two ways:
  1. adding a new layer (striping),
  2. using references.

- RDF and languages based on it allow one to use either striping or references interchangeably.

1.7 Namespaces

So far, all of the examples of XML documents used a single DTD. It is becoming much more common to use several DTDs in a single document. This has the important advantage that markup vocabulary that is already available can be reused rather than being invented again. However, simply merging the vocabularies of multiple DTDs can have undesirable consequences, such as:

- The same term can be used in different ways. For example, "locus" is an attribute in the Bioinformatic Sequence Markup Language (BSML), but it is an element in BioML.
- The same term can have different meanings. This is especially true of commonly occurring terms such as "value" and "label."
- The same term might have the same use and meaning, but it may be constrained differently. For example, the "Sequence" element occurs in several DTDs and has the same meaning, but the content and attributes that are allowed will vary.

Namespaces were introduced to XML to allow one to use multiple DTDs or XML schemas without confusing the names of elements and attributes that have more than one meaning. A *namespace* is a URI that serves as means of distinguishing a set of terms. For example, `reaction` is used both in the Systems Biology Markup Language (SBML) (SBML 2003) and in CML. The SBML namespace is `http://www.sbml.org/sbml/level2`. The CML namespace dealing with chemical reaction terminology is `http://www.xml-cml.org/schema/cml2/react`. By using the namespaces one can ensure that any use of `reaction` is unambiguous.

Within an XML document namespaces are specified using an abbreviation called the *namespace prefix*. For example, if one wishes to use both CML and SBML reactions in the same document, then one must declare prefixes as follows:

```
xmlns:cmlr="http://www.xml-cml.org/schema/cml2/react"
xmlns:sbml="http://www.sbml.org/sbml/level2"
```

These declarations are attributes that can be added to any element, but they are most commonly added to the root element. Once the prefixes have been declared, one can use the prefixes for elements and for attributes. For example, the following document mixes CML, BioML and SBML terminology:

```
<bioml:organism
 xmlns:cml="http://www.xml-cml.org/schema/cml2/core"
 xmlns:cmlr="http://www.xml-cml.org/schema/cml2/react"
 xmlns:bioml="http://xml.coverpages.org/bioMLDTD-19990324.txt"
 xmlns:sbml="http://www.sbml.org/sbml/level2"
>
  <bioml:species>Homo sapiens</bioml:species>
  <sbml:reaction sbml:id="reaction_1" sbml:reversible="false">
    <sbml:listOfReactants>
      <sbml:speciesReference sbml:species="X0"/>
    </sbml:listOfReactants>
    <sbml:listOfProducts>
      <sbml:speciesReference sbml:species="S1"/>
    </sbml:listOfProducts>
  </sbml:reaction>
  <cmlr:reaction>
    <cmlr:reactantList>
      <cml:molecule cml:id="r1"/>
    </cmlr:reactantList>
    <cmlr:productList>
      <cml:molecule cml:id="p1"/>
    </cmlr:productList>
  </cmlr:reaction>
  ...
</bioml:organism>
```

There are several ambiguities in the document above. As we have already noted, CML and SBML both use `reaction`. The meanings are the same, but they are specified differently. For example, CML uses `reactantList` for what SBML calls `listOfReactants`. A more subtle ambiguity is the use of `species` by both SBML and BioML. Here the the meanings are different. In SBML a species is a chemical species. In BioML it is an organism species.

One can use any prefix to designate a namespace within an XML element. For example, one could have used `xyz` instead of `bioml` in the document above. However, it is better to use prefixes that clearly abbreviate the namespace URI. When an element name or attribute has a namespace prefix, it is said to be *qualified* by the namespace.

One can also declare a namespace to be the *default* namespace. When there is a default namespace, then unqualified element names belong to the default namespace. The example above could be simplified somewhat by using a default namespace as follows:

```
<organism
 xmlns="http://xml.coverpages.org/bioMLDTD-19990324.txt"
 xmlns:sbml="http://www.sbml.org/sbml/level2"
 xmlns:cml="http://www.xml-cml.org/schema/cml2/core"
 xmlns:cmlr="http://www.xml-cml.org/schema/cml2/react"
>
  <species>Homo sapiens</species>
  <sbml:reaction sbml:id="reaction_1" sbml:reversible="false">
    <sbml:listOfReactants>
      <sbml:speciesReference sbml:species="X0"/>
    </sbml:listOfReactants>
    <sbml:listOfProducts>
      <sbml:speciesReference sbml:species="S1"/>
    </sbml:listOfProducts>
  </sbml:reaction>
  ...
</organism>
```

It is important to note that the default namespace applies only to element names, not to attributes. Because of this limitation, many authors have chosen to avoid using default namespaces altogether and to explicitly qualify every element and attribute. This has the advantage that such documents are somewhat easier to read, especially when one is using more than two or three namespaces.

The namespace URI need not be the same as the location of the DTD or schema. For example, the CML core has the namespace `http://www.xml-cml.org/schema/cml2/core`, but the actual location of the schema is `www.xml-cml.org/dtdschema/cmlCore.xsd`. Consequently, for each namespace one needs to know the URI, the location and the most commonly used abbreviation. The namespaces that are the most important for ontologies are

bioml:	Biopolymer Markup Language	`http://xml.coverpages.org/bioMLDTD-19990324.txt`
cellml:	Cell Markup Language	`http://www.cellml.org/cellml/1.0`
cmeta:	Cell Meta Language	`http://www.cellml.org/metadata/1.0`
cml:	Chemical Markup Language	`http://www.xml-cml.org/schema/cml2/core`
dc:	Dublin Core Elements	`http://purl.org/dc/elements/1.1/`
dcterms:	Dublin Core Terms	`http://purl.org/dc/terms/`
go:	Gene Ontology	`http://ftp://ftp.geneontology.org/pub/go/xml/dtd/go.dtd`
mathml:	Mathematics Markup Language	`http://www.w3.org/1998/Math/MathML`
owl:	Web Ontology Language	`http://www.w3.org/2002/07/owl`
rdf:	RDF	`http://www.w3.org/1999/02/22-rdf-syntax-ns`
rdfs:	RDF Schema	`http://www.w3.org/2000/01/rdf-schema`
sbml:	Systems Biology Markup Language	`http://www.sbml.org/sbml/level2`
stm:	Technical Markup Language	`http://www.xml-cml.org/schema/stmml`
xmlns:	XML Namespaces	`http://www.w3.org/XML/1998/namespace`
xsd:	XML Schema (original)	`http://www.w3.org/2000/10/XMLSchema`
xsd:	XML Schema (proposed)	`http://www.w3.org/2001/XMLSchema`
xsi:	XML Schema instances	`http://www.w3.org/2001/XMLSchema-instance`
xsl:	XML Transform	`http://www.w3.org/1999/XSL/Transform`
xtm:	Topic Maps	`http://www.topicMaps.org/xtm/1.0/`

Summary

- Namespaces organize multiple vocabularies so that they may be used at the same time.
- Namespaces are URIs that are declared in an XML document.
- Each namespace is either the default namespace or it has an abbreviation called a prefix.
- Element names and attributes may be qualified by using the namespace prefix.
- The default namespace applies to all unqualified elements. It does not apply to unqualified attributes.

1.8 Exercises

1. A spreadsheet was exported in comma-delimited format. The first few lines look like this:

```
element_id,sequence_id,organism_name,seq_length,type
U83302,MICR83302,Colaptes rupicola,1047,DNA
U83303,HSU83303,Homo sapiens,3460,DNA
U83304,MMU83304,Mus musculus,51,RNA
U83305,MIASSU833,Accipiter striatus,1143,DNA
```

   Show how these records would be written as XML elements using the `bio_sequence` tag.

2. For the spreadsheet in exercise 1.1 above, show the corresponding XML DTD. The `element_id` attribute is a unique key for the element. Assume that all attributes are optional. The molecule type is restricted to the biologically significant types of biopolymer.

3. Here is a relational database table that defines some physical units:

name	prefix	unit	exponent
millisecond	milli	second	1
per_millisecond	milli	second	-1
millivolt	milli	volt	1
microA_per_mm2	micro	ampere	1
microA_per_mm2	milli	meter	-2
microF_per_mm2	micro	farad	1
microF_per_mm2	milli	meter	-2

A physical unit is, in general, composed of several factors. This was encoded in the relational table by using several records, one for each factor. The `microF_per_mm2` unit, for example, is the ratio of microfarads by square millimeters.

This relational database table illustrates how several distinct concepts can be encoded in a single relational table. In general, information in a relational database about a single concept can be spread around several records, and a single record can include information about several concepts. This can make it difficult to understand the meaning of a relational table, even when the relational schema is available.

Show how to design an XML document so that the information about the two concepts (i.e, the physical units and the factors) in the table above are separated.

4. This next relational database table defines some of the variables used in the Fitzhugh-Nagumo model (Fitzhugh 1961; Nagumo 1962) for the transmission of signals between nerve axons:

component	variable	initial	physical_unit	interface
membrane	u	-85.0	millivolt	out
membrane	Vr	-75.0	millivolt	out
membrane	Cm	0.01	microF_per_mm2	
membrane	time		millisecond	in
ionic_current	I_ion		microA_per_mm2	out
ionic_current	v			in
ionic_current	Vth		millivolt	in

The physical units are the ones defined in exercise 3 above. Extend the solution of that exercise to include the data in the table above. Note that once again, multiple concepts have been encoded in a single relational database table. This exercise is based on an example on the CellML website (CellML 2003).

5. Use an XML editor (such as Xerlin or XML Spy) to construct the examples in the previous two exercises. Follow these steps:

   (a) Cut and paste the following DTD into a file:

   ```
   <?xml version="1.0">
   <!DOCTYPE model [
     <!ELEMENT model (physical_unit*,component*)>
     <!ELEMENT physical_unit (factor)*>
     <!ATTLIST physical_unit name ID #REQUIRED>
     <!ELEMENT factor EMPTY>
     <!ATTLIST factor
          prefix   CDATA #IMPLIED
          unit     CDATA #REQUIRED
          exponent CDATA "1">
     <!ELEMENT component (variable)*>
     <!ATTLIST component name ID #REQUIRED>
     <!ELEMENT variable EMPTY>
     <!ATTLIST variable
         name          CDATA    #REQUIRED
         initial       CDATA    #IMPLIED
         physical_unit IDREF    "dimensionless"
         interface     (in|out) #IMPLIED>
   ]>
   <model/>
   ```

   (b) Open the file with your XML editor.

   (c) Create the elements and enter the attributes shown in the two database tables in the two previous exercises.

   (d) Save the file, and open it with an ordinary text editor.

   (e) Verify that the resulting file has the data as shown in the answers to the exercises above.