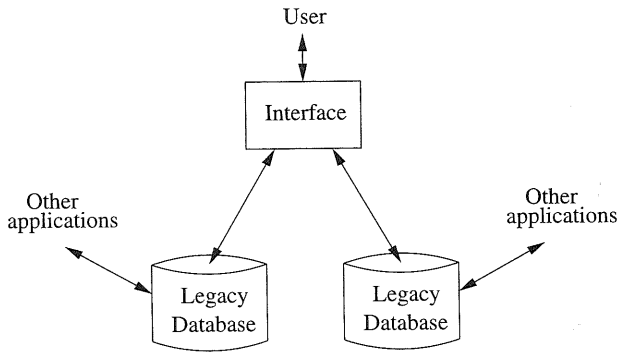

Figure 11.2: Integrating two legacy databases through an interface that supports semistructured data

Example 11.3: We can see in Fig. 11.1 a possible effect of information about stars being gathered from several sources. Notice that the address information for Carrie Fisher has an address concept, and the address is then broken into street and city. That situation corresponds roughly to data that had a nested-relation schema like `Stars(name, address(street, city))`.

On the other hand, the address information for Mark Hamill has no address concept at all, just street and city. This information may have come from a schema such as `Stars(name, street, city)` that can represent only one address for a star. Some of the other variations in schema that are not reflected in the tiny example of Fig. 11.1, but that could be present if movie information were obtained from several sources, include: optional film-type information, a director, a producer or producers, the owning studio, revenue, and information on where the movie is currently playing. □

11.1.4 Exercises for Section 11.1

Exercise 11.1.1: Since there is no schema to design in the semistructured-data model, we cannot ask you to design schemas to describe different situations. Rather, in the following exercises we shall ask you to suggest how particular data might be organized to reflect certain facts.

a) Add to Fig. 11.1 the facts that *Star Wars* was directed by George Lucas and produced by Gary Kurtz.

b) Add to Fig. 11.1 information about *Empire Strikes Back* and *Return of the Jedi*, including the facts that Carrie Fisher and Mark Hamill appeared in these movies.

c) Add to (b) information about the studio (Fox) for these movies and the address of the studio (Hollywood).

Exercise 11.1.2: Suggest how typical data about a genealogy, as was described in Exercise 4.1.6, could be represented in the semistructured model.

Exercise 11.1.3: Suggest how typical data about banks and customers, as in Exercise 4.1.1, could be represented in the semistructured model.

Exercise 11.1.4: Suggest how typical data about players, teams, and fans, as was described in Exercise 4.1.3, could be represented in the semistructured model.

! **Exercise 11.1.5:** UML and the semistructured-data model are both "graphical" in nature, in the sense that they use nodes, labels, and connections among nodes as the medium of expression. Yet there is an essential difference between the two models. What is it?

11.2 XML

XML (*Extensible Markup Language*) is a tag-based notation designed originally for "marking" documents, much like the familiar HTML. Nowadays, data with XML "markup" can be represented in many ways. However, in this section we shall refer to XML data as represented in one or more documents. While HTML's tags talk about the presentation of the information contained in documents — for instance, which portion is to be displayed in italics or what the entries of a list are — XML tags are intended to talk about the meanings of pieces of the document.

In this section we shall introduce the rudiments of XML. We shall see that it captures, in a linear form, the same structure as do the graphs of semistructured data introduced in Section 11.1. In particular, tags can play the same role as the labels on the arcs of a semistructured-data graph.

11.2.1 Semantic Tags

Tags in XML are text surrounded by triangular brackets, i.e., `<...>`, as in HTML. Also as in HTML, tags generally come in matching pairs, with an *opening tag* like `<Foo>` and a matched *closing* tag that is the same word with a slash, like `</Foo>`. Between a matching pair `<Foo>` and `</Foo>`, there can be text, including text with nested HTML tags, and any number of other nested matching pairs of XML tags. A pair of matching tags and everything that comes between them is called an *element*.

A single tag, with no matched closing tag, is also permitted in XML. In this form, the tag has a slash before the right bracket, for example, `<Foo/>`. Such a tag cannot have any other elements or text nested within it. It can, however, have attributes (see Section 11.2.4).

There is a small matter that values of attributes and elements can have different types, e.g., integers or strings, while relational attributes each have a unique type. We could treat the two attributes named `value` as always being strings, and interpret those strings that were integers or another type properly as we processed the data. Or we could split each of the last two relations into as many relations as there are different types of data.

11.2.8 Exercises for Section 11.2

Exercise 11.2.1: Repeat Exercise 11.1.1 using XML.

! **Exercise 11.2.2:** How would you represent an empty element (one that had neither text nor subelements) in the database schema of Section 11.2.7?

! **Exercise 11.2.3:** In Section 11.2.7 we gave a database schema for representing documents that do not have *mixed content* — elements that contain a mixture of text (`#PCDATA`) and subelements. Show how to modify the schema when elements can have mixed content.

Exercise 11.2.4: Show that any relation can be represented by an XML document. *Hint*: Create an element for each tuple with a subelement for each component of that tuple.

11.3 Document Type Definitions

For a computer to process XML documents automatically, it is helpful for there to be something like a schema for the documents. It is useful to know what kinds of elements can appear in a collection of documents and how elements can be nested. The description of the schema is given by a grammar-like set of rules, called a *document type definition*, or DTD. It is intended that companies or communities wishing to share data will each create a DTD that describes the form(s) of the data they share, thus establishing a shared view of the semantics of their elements. For instance, there could be a DTD for describing protein structures, a DTD for describing the purchase and sale of auto parts, and so on.

11.3.1 The Form of a DTD

The gross structure of a DTD is:

```
<!DOCTYPE root-tag [
    <!ELEMENT element-name (components)>
        more elements
]>
```

are the ID's that will appear on lists that are the values of `starredIn` elements. Symmetrically, the attribute `starsOf` of `Movie` is an `IDREFS`, a list of ID's for stars. □

11.3.5 Exercises for Section 11.3

Exercise 11.3.1: Add to the document of Fig. 11.10 the following facts:

a) Matt Damon starred in *The Departed* (2006).

b) Carrie Fisher and Mark Hamill also starred in *The Empire Strikes Back* (1980) and *Return of the Jedi* (1983).

c) Harrison Ford also starred in *Star Wars*, in the two movies mentioned in (a), and the movie *Air Force One* (1997).

d) Carrie Fisher also starred in *Charlies Angels: Full Throttle* (2003).

Exercise 11.3.2: Suggest how typical data about a genealogy, as was described in Exercise 4.1.6, could be represented as a DTD.

Exercise 11.3.3: Suggest how typical data about banks and customers, as was described in Exercise 4.1.1, could be represented as a DTD.

Exercise 11.3.4: Suggest how typical data about players, teams, and fans, as was described in Exercise 4.1.3, could be represented as a DTD.

! **Exercise 11.3.5:** Using your representation from Exercise 11.2.4, devise an algorithm that will take any relation schema (a relation name and a list of attribute names) and produce a DTD describing a document that represents that relation.

11.4 XML Schema

XML Schema is an alternative way to provide a schema for XML documents. It is more powerful than DTD's, giving the schema designer extra capabilities. For instance, XML Schema allows arbitrary restrictions on the number of occurrences of subelements. It allows us to declare types, such as integer or float, for simple elements, and it gives us the ability to declare keys and foreign keys.

11.4.1 The Form of an XML Schema

An XML Schema description of a schema is itself an XML document. It uses the namespace at the URL:

```
http://www.w3.org/2001/XMLSchema
```

capability is similar to what we get with ID's and IDREF's in a DTD (see Section 11.3.4). However, the latter are untyped references, while references in XML Schema are to particular types of elements. The form of a foreign-key definition in XML Schema is:

```
<xs:keyref name = foreign-key name refer = key name >
  <xs:selector xpath = path description >
  <xs:field xpath = path description >
</xs:keyref>
```

The schema element is `xs:keyref`. The foreign-key itself has a name, and it refers to the name of some key or unique value. The selector and field(s) are as for keys.

Example 11.20: Figure 11.20 shows the definition of an element `<Stars>`. We have used the style of XML Schema where each complex type is defined within the element that uses it. Thus, we see at lines (4) through (6) that a `<Stars>` element consists of one or more `<Star>` subelements.

At lines (7) through (11), we see that each `<Star>` element has three kinds of subelements. There is exactly one `<Name>` and one `<Address>` subelement, and any number of `<StarredIn>` subelements. In lines (12) through (15), we find that a `<StarredIn>` element has no subelements, but it does have two attributes, `title` and `year`.

Lines (22) through (26) define a foreign key. In line (22) we see that the name of this foreign-key constraint is `movieRef` and that it refers to the key `movieKey` that was defined in Fig. 11.19. Notice that this foreign key is defined within the `<Stars>` definition. The selector is `Star/StarredIn`. That is, it says we should look at every `<StarredIn>` subelement of every `<Star>` subelement of a `<Stars>` element. From that `<StarredIn>` element, we extract the two fields `title` and `year`. The `@` indicates that these are attributes rather than subelements. The assertion made by this foreign-key constraint is that any title-year pair we find in this way will appear in some `<Movie>` element as the pair of values for its subelements `<Title>` and `<Year>`. □

11.4.8 Exercises for Section 11.4

Exercise 11.4.1: Write the XML Schema definitions of Fig. 11.19 and 11.20 as a DTD.

Exercise 11.4.2: Give an example of a document that conforms to the XML Schema definition of Fig. 11.12 and an example of one that has all the elements mentioned, but does not conform to the definition.

Exercise 11.4.3: Rewrite Fig. 11.12 so that there is a named complex type for `Movies`, but no named type for `Movie`.

```
1)  <? xml version = "1.0" encoding = "utf-8" ?>
2)  <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">

3)  <xs:element name = "Stars">

4)     <xs:complexType>
5)        <xs:sequence>
6)           <xs:element name = "Star" minOccurs = "1"
                   maxOccurs = "unbounded">
7)              <xs:complexType>
8)                 <xs:sequence>
9)                    <xs:element name = "Name"
                         type = "xs;string" />
10)                   <xs:element name = "Address"
                         type = "xs:string" />
11)                   <xs:element name = "StarredIn"
                            minOccurs = "0"
                            maxOccurs = "unbounded">
12)                      <xs:complexType>
13)                         <xs:attribute name = "title"
                               type = "xs:string" />
14)                         <xs:attribute name = "year"
                               type = "xs:integer" />
15)                      </xs:complexType>
16)                   </xs:element>
17)                </xs:sequence>
18)             </xs:complexType>
19)          </xs:element>
20)       </xs:sequence>
21)    </xs:complexType>

22)    <xs:keyref name = "movieRef" refers = "movieKey">
23)       <xs:selector xpath = "Star/StarredIn" />
24)       <xs:field  xpath = "@title" />
25)       <xs:field  xpath = "@year" />
26)    </xs:keyref>

27) </xs:element>
```

Figure 11.20: Stars with a foreign key