Chapter 2

DATA MODELS FOR SEMISTRUCTURED DATA

Traditionally, real world semantics are captured in a data model, and mapped to the database schema. The real world semantics are modeled as constraints and used to ensure consistency of the data in the resulting database. Similarly, in semistructured databases the consistency of the data can be enforced through the use of constraints. There are two approaches to designing a schema for a semistructured database. The first follows the traditional approach and captures the real world constraints in a data model. The second approach is used in the case where a semistructured document exists without a schema. Following this approach the constraints are extracted from the document and modeled using a data model.

A data model that is used in the design of schemas for semistructured data has different requirements than those used in the design of schemas for relational databases. In order to support the second approach outlined above, the data model must provide a way to model the document instance, the document schema, and identifying attributes of element sets. The fundamental concepts of semistructured data must also be part of the model. They include the hierarchical structure of element sets, and ordering of element sets and attributes. The model must also be able to represent constraints that are needed in the design of schemas such as binary and n-ary relationship sets, participation constraints of element sets in relationship sets, attributes and element sets, and attributes of relationship sets.

Table 2.1 gives a summary of the concepts that are important in a data model for semistructured data. The exact meaning of these concepts will be uncovered in later sections of this chapter and the reason we have chosen this particular set of requirements will be explained in subsequent chapters in this book.

The following is a running example that is based on the XML document in Figure 2.1. We use the term *element* to describe a particular element and a tag

Concepts of Semistructured Data Model
document instance
document schema
identifier of element sets
hierarchical structure of element sets
binary and n-ary relationship sets
participation constraints of element sets in relationship sets
references between element sets
attributes and element sets
attributes of relationship sets
ordering of element sets and attributes

Table 2.1. Essential concepts of a data model for semistructured data

name in a document, and the term *element set* to describe a set of elements with the same tag name in a document. Similarly, we use the term *relationship* to describe a relationship between two elements in a document and the term *relationship set* to describe a set of relationships which relate instances of the same element sets.

Example 2.1 *In the XML document in Figure 2.1, there are element sets department, course, title, student, stuName, address, hobby and grade. Elements are instances of the element sets, so there is a course element that has an attribute code with value "CS1102", and another course element that has an attribute code with value "CS2104".*

The nesting of the element sets forms the hierarchical structure of the document, e.g. course is nested within department, student is nested within course, and so on. We say there is a relationship set between department and course, and a relationship set between course and student. Relationships are instances of relationship sets, so there is a relationship between element department that has an attribute name with value "CS" and element course that has an attribute code with value "CS1102".

Element sets have attributes, e.g. name is an attribute of department, and code is an attribute of course.

In the following sections, we will survey some of the main data models that have been proposed for semistructured documents, such as DTD and DOM, and compare them.

2.1 Document Type Definition

The Document Type Definition (DTD) language [Bray et al., 2000] and other schema definition languages, such as XML Schema [Thompson et al.,

```
<enrollment>
  <department name="CS">
    <course code="CS1102">
      <title>Data Structure</title>
      <student stuNo="stu123">
        <stuName> Zheng Zhang </stuName>
        <address>hostel 01–03–A </address>
        <grade>A</grade>
      </student>
      <student stuNo="stu125">
        <stuName> Liang Chen </stuName>
        <hobby>hockey</hobby>
        <hobby>reading</hobby>
        <grade>B</grade>
      </student>
    </course>
    <course code="CS2104">
      <student stuNo="stu123">
        <stuName> Zheng Zhang </stuName>
        <address>hostel 01–03–A</address>
      </student>
      <student stuNo="stu125">
        <stuName> Liang Chen </stuName>
        <hobby>hockey</hobby>
        <hobby>reading</hobby>
      </student>
    </course>
  </department>
</enrollment>
```

Figure 2.1. Example XML document

2001] and RELAXNG [ISO/IEC, 2000], have become a familiar way to represent the schema of an XML document. The DTD language uses regular expressions to describe the schema. In the DTD language, it is possible to represent element sets, the hierarchical structure of element sets, and some constraints on the element sets, attributes, and relationship sets. We investigate how these concepts are represented in the DTD in this section.

In a DTD, the participation constraint on a child element set in a relationship set is stated explicitly using the symbols ?, +, * which represent zero-to-one occurrences (written as $0:1$), one-to-many occurrences (written as $1:n$), and zero-to-many occurrences (written as $0:n$) respectively. Element sets either form a sequence (that is, there is an ordering specified on them) or they are disjunctive (that is, one or other of them occurs). An attribute can be tagged as an identifier, indicating that it is expected to have a unique value within an instance XML document. An attribute can have a string value or be a reference

```
<!ELEMENT enrollment    (department+)>
<!ELEMENT department    (course+)>
    <!ATTLIST department  name  ID  #REQUIRED>
<!ELEMENT  course   (title?, student*)>
    <!ATTLIST course  code  ID   #REQUIRED>
<!ELEMENT title   (#PCDATA)>
<!ELEMENT student   (stuName, address?, hobby*, grade?)>
    <!ATTLIST student  stuNo  #REQUIRED>
<!ELEMENT stuName   (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT hobby   (#PCDATA)>
<!ELEMENT grade   (#PCDATA)>
```

Figure 2.2. A DTD for the document in Figure 2.1

to the identifying attribute of an element set. For attributes it is possible to specify if they are required, optional, have a default value or have a fixed value.

Example 2.2 *Consider the DTD in Figure 2.2 for the document in Figure 2.1. The hierarchical structure is represented in the nesting of the element sets. For example, the second line in Figure 2.2 states that the element set course is a subelement of the element set department. The second line also specifies the participation constraint on the element set course in the relationship set between department and course, namely, that there can be one or more courses in each department (indicated by the "+").*

The third line of the DTD shows that element set department has an attribute name. The keyword "#REQUIRED" indicates that the attribute name must appear in every department, while the keyword "ID" indicates that the value of the attribute is unique within the XML document. That is, there is only one department with any particular name in this document.

The following two lines show that the element set course has subelement sets title and student. They occur as a sequence in the order specified, and every course has an optional title and zero or more students. The keyword "#PCDATA" indicates that element set title is a leaf element set, that is it has no subelement sets and instead elements belonging to this set have a value.

The last six lines describe the element set student, which has subelement sets stuName, address, hobby and grade, and attribute stuNo. Although the attribute stuNo is an identifier in the usual sense, it is not represented as an ID attribute in the DTD because the same student can take many courses, and thus, there will be many student elements with the same value for stuNo as demonstrated in the XML document in Figure 2.1.

The schema described in the DTD in Figure 2.2 represents the structure of the XML document in Figure 2.1. However, there is a problem with this

```
<!ELEMENT enrollment     (department+, student+)>
<!ELEMENT department     (course+)>
    <!ATTLIST department  name  ID  #REQUIRED>
<!ELEMENT course   (title?, stuRef*)>
    <!ATTLIST course  code  ID   #REQUIRED>
<!ELEMENT title   (#PCDATA)>
<!ELEMENT student   (stuName, address?, hobby*)>
    <!ATTLIST student  stuNo  ID #REQUIRED>
<!ELEMENT stuName   (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT hobby   (#PCDATA)>
<!ELEMENT stuRef    (grade?)>
    <!ATTLIST stuRef  stuNo IDREF  #REQUIRED>
<!ELEMENT grade   (#PCDATA)>
```

Figure 2.3. A DTD for the document in Figure 2.1 without replication

schema. Data is replicated in the instance, for example, the details of each student are repeated for every course the student takes. This replication of information can be avoided if the structure of the XML document is changed.

Example 2.3 *Figure 2.3 shows a possible schema that does not exhibit data replication. In this schema, student is no longer nested within course. The element set student is now nested within enrollment and there is a reference from course to student. Based on real world semantics, we know that grade represents the grade of a student within a course. Thus, in Figure 2.3, grade is more correctly represented as an attribute of the relationship (or reference) between course and student.*

Let us take a closer look at the DTD in Figure 2.3. Element set department has subelement set course. Element set course has subelement sets title and stuRef. Element set stuRef has a subelement set grade and an attribute stuNo. The attribute stuNo is like a foreign key, referring to a student with a particular stuNo. Notice that there will only be one element for each student in an XML document, so attribute stuNo can be an ID attribute of element set student.

We now consider how well the DTD language supports the requirements of a data model for designing a schema for a semistructured document. The DTD describes only the schema and does not describe an instance of the document. The hierarchical structure of element sets is supported well but the only relationship sets that can be described directly are those within the hierarchical structure.

Example 2.2 illustrates one of the problems that arises through only being able to directly support the hierarchical relationship. Relationships that are not

hierarchical relationships can be modeled using references. Similarly relationships of degree n where $n > 2$ can be modeled using references. However, without a direct way of supporting these kinds of relationships, valuable semantic information is lost.

Even when the DTD is small and not very complex, as shown in Figure 2.2, it is difficult to quickly gain an idea of the structure of the data without looking at the details closely.

Participation constraints on children in a relationship set are represented directly. For example in Figure 2.2, a course has zero to many students. However it is not possible to express participation constraints on parents of a relationship set. For example, we cannot indicate that a student can take many courses.

The concepts of element sets and attributes follow the same concepts in XML documents, which differs from the concepts in data modeling. In data modeling, an attribute is a property of an element set, but in XML such properties can be represented using either attributes or element sets. For example, the element set *stuName* which is represented as a subelement of *student* in Figure 2.3 would normally be considered an attribute in data modeling.

It is not possible to directly distinguish between attributes of element sets and attributes of relationship sets. For example in Figure 2.2, the element set *grade* represents the grade a student scored in a particular course and should be considered an attribute of the relationship set between element sets *course* and *student,* but it is represented in the same way that an attribute of an object set is represented, for example it is represented in the same way that element set *stuName,* which is simply an attribute of element *student,* is represented. In Figure 2.3, a new element set *stuRef* is introduced specifically for the purpose of representing the *grade* as related to the relationship between *course* and *student.* Although the new element set *stuRef* removes redundancy, it is still not possible to show that *grade* is related to the relationship between course and student.

It is possible to specify an ordering on subelement sets. In fact this ordering is possibly stricter than required since it is not easy to specify a group of subelements where ordering does not matter, which is often what we would like to represent. For example, in Example 2.2 the subelements of any instance of *student,* namely *stuName, address, hobby* and *grade* are expected to appear in that order but from a data modeling perspective we are not concerned with the ordering of these subelements.

2.2 DOM, OEM and DataGuide

Some other data models that are commonly used to depict XML documents and their structure are DOM (Document Object Model), OEM (Object Exchange Model), and DataGuide.

Document Object Model

The DOM (Document Object Model) [Apparao and Byrne, 1998] depicts the instance of an XML document as a tree. Each node represents an object that contains one of the components from an XML structure. The three most common type nodes are element nodes, attribute nodes and text nodes.

As illustrated in Figure 2.4, *text nodes* have no name but carry text (e.g. the text node with text *"Data Structure"); attribute nodes* have both a name and carry text (e.g. the attribute node with attribute name *name* and text value "CS"); and *element nodes* have a name and may have children (e.g. the element node with element name *course*). The edges between nodes represent the relationships between the nodes.

How well does the DOM support the requirements of a data model for designing a schema for a semistructured document? A DOM tree represents the instance of a document, showing the hierarchical structure of the elements, and the implicit relationships between the elements due to the hierarchical structure. It is possible to distinguish between attributes and elements. However, because the DOM represents an instance of an XML document, it does not represent schema information directly, such as the degree of relationship sets, and participation constraints on element sets in relationship sets. For the same reason it is not possible to distinguish between ordered elements and unordered elements, or whether an attribute belongs to a relationship set or an element set.

Object Exchange Model

The Object Exchange Model (OEM) [McHugh et al., 1997] also depicts the contents of an XML document. An OEM model is a labeled directed graph where the vertices are objects, and the edges are relationships. Figure 2.5(a) shows the OEM model for the XML document in Figure 2.1.

Each object has a unique object identifier (OID), a label and a value. There are two types of objects, atomic and complex. Both atomic and complex objects are depicted as 3-tuples: (OID, label, value). An atomic object contains a value from one of the disjoint basic atomic types, e.g., integers, real, string, etc. A complex object is a composition of objects where the value of a complex object is a set of object references, denoted as a set of (label, OID) pairs. We illustrate these concepts in Example 2.4.

Example 2.4 *Consider the OEM model in Figure 2.5(a). The leaf nodes of the graph are the atomic objects and the internal nodes are the complex objects. The complex object with object identifier* &1 *and name department is specified in the 3-tuple:*

$$(\&1, department, \{(name, \&2), (course, \&4), (course, \&5)\}),$$

where the set of tuples represent the objects that object &1 *references.*

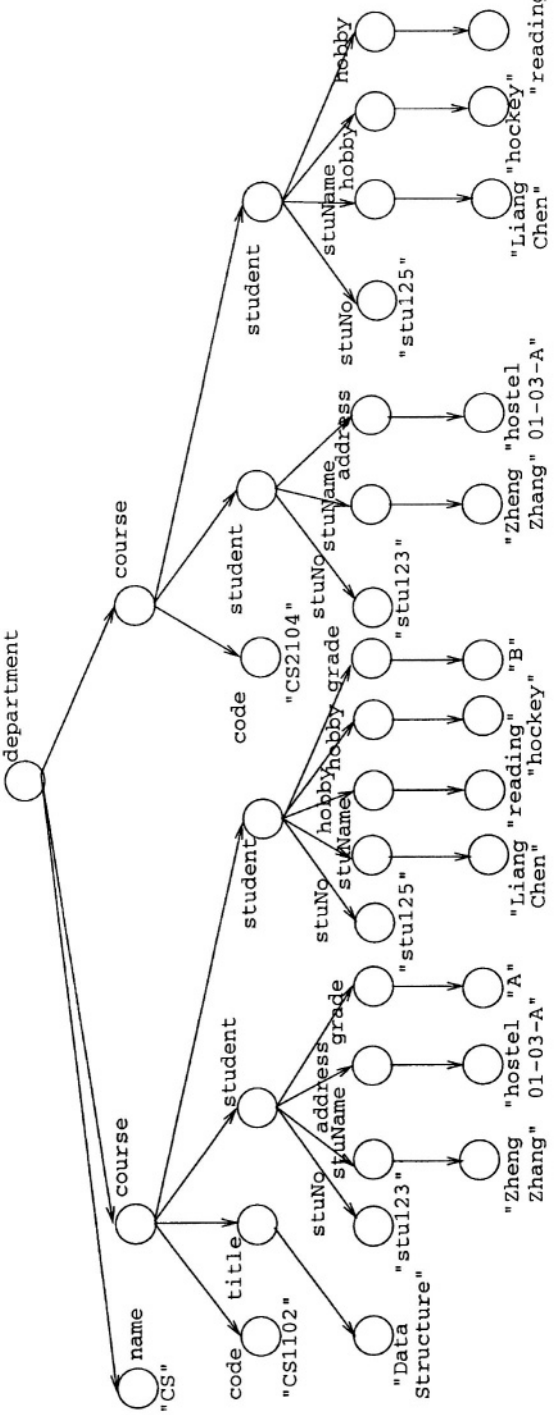

Figure 2.4. A DOM tree for the document in Figure 2.1

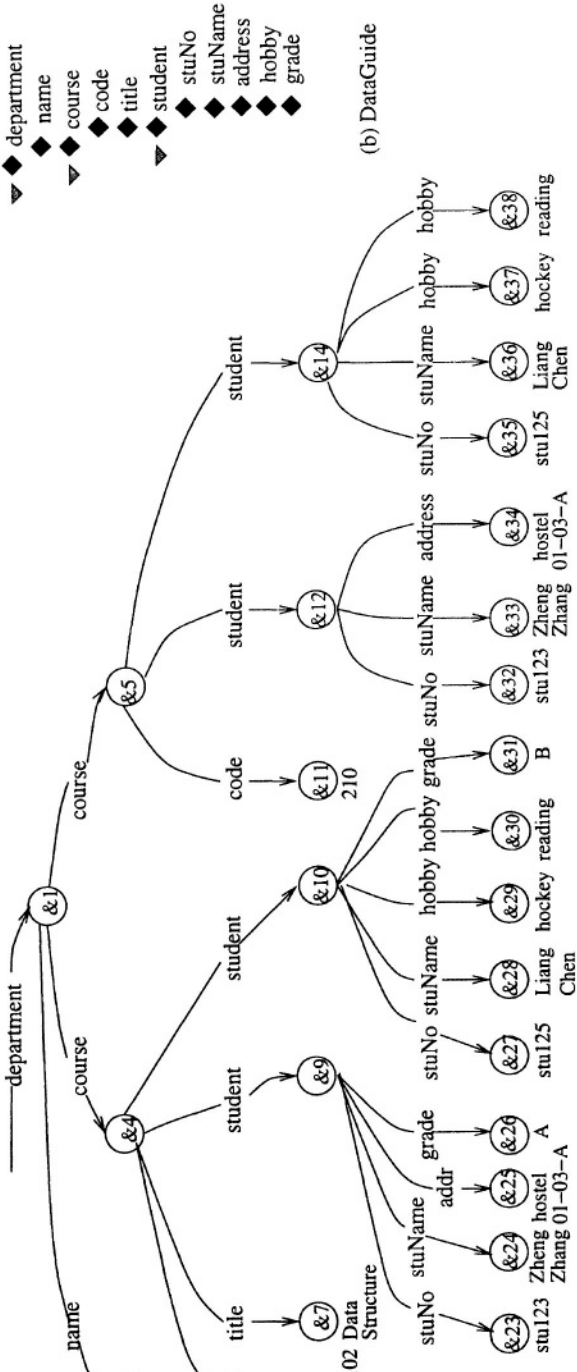

Figure 2.5. An (a) OEM diagram and its (b) DataGuide for the document in Figure 2.1

The three atomic objects with object identifiers &27, &28 *and* &31 *are specified in the following three 3-tuples respectively showing their OID, name and value:*

(&27, *stuNo, stu*125)
(&28, *stuName, Liang Chen*)
(&31, *grade, B*)

An OEM indicates the hierarchical structure of the objects. Although it has both the diagrammatic and textual representation, not only does it have the same shortcomings as the DOM but it also suffers from not distinguishing between elements and attributes.

DataGuide

A DataGuide [Goldman and Widom, 1997] models the schema of an OEM instance graph, depicting every path through the instance only once. Figure 2.5 shows the OEM model and its DataGuide for the XML document in Figure 2.1. From the DataGuide, it is easy to see that instances of the element sets *department, course* and *student* are complex objects (depicted by a triangle), where an instance of the element set *department* is composed of the atomic object *name,* and the multiple occurrence complex object *course,* where instances of the element set *course* in turn are composed of the atomic objects *code* and *title,* and the multiple occurrence complex object *student.*

How well does OEM with DataGuides support the requirements of a data model for designing a schema for a semistructured document? From this example, you can see that a DataGuide depicts only the hierarchical structure of the element sets and like the OEM it does not distinguish between element sets and attributes. It is in fact less expressive than the DTD since it is not possible to represent the participation constraints on element sets in relationship sets, and because there is no distinction between element sets and attributes it is not possible to represent the constraints on attributes that can be modeled in the DTD. It is not possible to represent references using OEM and DataGuides, which means it is not possible to model ID, IDREF and IDREFS from the DTD.

2.3 S3-graph

A Semi-Structured Schema Graph (S3-Graph) is a directed graph where each node in the graph can be classified into an *entity node* or a *reference node.* An entity node represents an entity which can be of basic atomic data type such as string, date or complex object type such as student. The former is also known as a *leaf entity node.* A reference node is a node which references to another entity node.

Each directed edge in the graph is associated with a *tag*. The tag represents the relationship between the source node and the destination node. The tag may be suffixed with a "*". The interpretations of tag and the suffix depend on the type of edge. There are three types of edges:

1 Component Edge

A node V_1 is connected to another node V_2 via a *component edge* with a tag T if V_2 is a component of V_1. This edge is denoted by a solid arrow line. If T is suffixed with a "*", the relationship is interpreted as "The entity type represented by V_1 has many T". Otherwise, the relationship is interpreted as "The entity type represented by V_1 has at most one T".

2 Referencing Edge

A node V_1 is connected to another node V_2 via a *referencing edge* if V_1 references the entity represented by node V_2. This type of edge is denoted by a dashed arrow line.

3 Root Edge

A node V_1 is pointed to by a *root edge* with a tag T if the entity type represented by V_1 is owned by the database. This edge is denoted by a solid arrow line without any source node for the edge, and there is no suffix for the tag T. In fact, V_1 is a *root node* in the S3-Graph.

Some roles R can be associated with a node V if there is a directed *(component or referencing)* edge pointing to V with tag R after removing any suffix "*".

Example 2.5 *Figure 2.6 shows the S3-Graph for the XML document in Figure 2.1. Node #1 represents an entity node, which represents the entity DEPARTMENT. This is also a root node. This node is associated with the role "department".*

Node #2 is another entity node of which database instance holds a string representing the NAME of a department. It is associated with the role "name ", and it is also a leaf node associated the atomic data type "string". Hence, any "NAME" data is of string type. The directed edge between node #1 and node #2 represents "Each DEPARTMENT has at most one NAME".

Nodes #3 and #6 are entity nodes which represents the entities COURSE and STUDENT which are complex object types. A complex object type such as COURSE is connected to leaf entity nodes #4 and #5 that are associated with the roles "code" and "title" respectively.

Note that the tag on the edge from node #1 to node #3 is suffixed with a "". Hence, the relationship is interpreted as "A DEPARTMENT has many COURSE".*

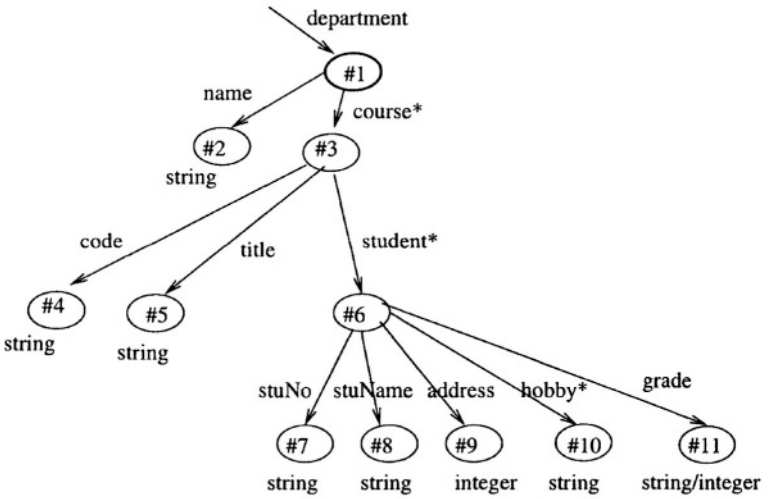

Figure 2.6. An S3-Graph for the document in Figure 2.1

How well does S3-Graph support the requirements of a data model for designing a schema for a semistructured document? We observe that S3-Graph captures the hierarchical structure of the element sets and provides for references. However, it does not distinguish between the attributes of entity types and relationship sets, e.g. it is not clear from the S3-Graph in Figure 2.6 that *grade* is an attribute of the relationship between *course* and *student.* Further, the S3-Graph is able to represent one-to-one and one-to-many binary relationship sets, and not ternary relationship sets.

2.4 CM Hypergraph and Scheme Tree

A data model that consists of two diagrams, the CM (conceptual-model) hypergraph and the scheme tree, was defined in [Embley and Mok, 2001]. The data model was designed to represent the semantics needed when devising algorithms that ensure the development of XML documents with "good" properties.

We will adopt the authors' term "object sets" when referring to "element sets" in this section. The CM hypergraph models the data conceptually, modeling object sets and relationship sets, providing a way to represent some participation constraints and the generalization relationship. The scheme tree models only the hierarchical structure of the document.

In the CM hypergraph, object sets are represented as labeled rectangles, e.g. the object set *department.* Relationship sets are represented by edges, and participation constraints are represented using arrow heads and the symbol "o" on

the edges. An edge with no arrow heads represents a many-to-many relationship set, an edge with one arrow head represents a many-to-one relationship, and an edge with an arrow head at both ends represents a one-to-one relationship. The symbol "o" indicates that an object is optional.

Another way to view the arrow head notation is as representing functional dependencies. From the arrow heads, we can derive that *code* $\rightarrow$ *title* and that *stuNo* $\rightarrow$ {*name, address*}.

The scheme tree represents the same information that a DataGuide represents, namely the hierarchical structure of the object sets. The edges represent the element-subelement relationship. An algorithm that generates the scheme tree from the CM hypergraph is described in [Embley and Mok, 2001]. Because the CM hypergraph is more expressive than the scheme tree, it is not possible to regenerate the CM hypergraph from the scheme tree.

Example 2.6 *Consider the CM hypergraph and scheme tree in Figure 2.7. The CM hypergraph in Figure 2.7(a) has object sets department, name, course, code, title, student, stuNo, stuName, address, grade and hobby. The CM hypergraph succinctly models the following constraints. A department has only one name and one or more courses. The name is unique. A course belongs to only one department, has a unique code and an optional title.*

The edge between object sets department and name indicates a one-to-one relationship. The edge between department and course indicates a one-to-many relationship. The edge between student and hobby indicates a many-to-many relationship between student and hobby where hobby is optional.

There is a ternary relationship set among course, student and grade. Each course, student pair has only one grade. A student has a unique stuNo, a stuName, an optional address, and zero or more hobbies.

The scheme tree in Figure 2.7(b) represents the hierarchical structure, with department and name at the root; course with code and title are nested within department; student is nested within course; and grade is nested within student. The student information, stuNo, stuName, address, hobby, forms a separate scheme tree.

How well do CM hypergraphs and scheme trees support the requirements of a data model for designing a schema for semistructured data? This data model represents a conceptual model (in the CM hypergraph) and the hierarchical structure (in the scheme tree) of the schema. It is not possible to represent an instance of a document in this data model.

CM hypergraphs can model both binary and n-ary relationships (where $n > 2$) with the cardinality of the object sets taking part in the relationships. Notice that the hierarchical nesting is not modeled in the CM hypergraph directly. Since CM hypergraphs do not distinguish between attributes and object sets, the number of object sets in a CM hypergraph quickly becomes very large and

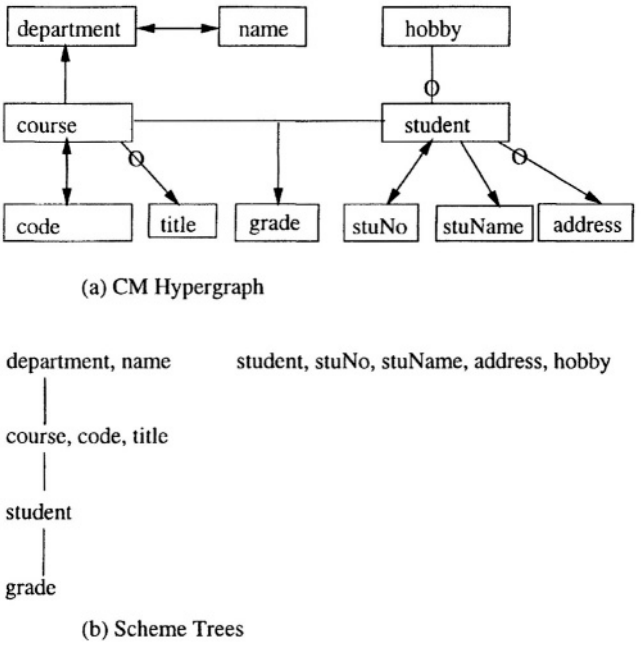

Figure 2.7. A CM Hypergraph and Scheme Tree for the schema in Figure 2.3

the graph very complex. One of the advantages of the ER diagram is that it is possible to have two levels of representation, one without attributes and one with all the attributes. The two levels of representation are not possible with CM hypergraphs, as there is no concept of attribute. Because CM hypergraphs are unable to represent the hierarchical relationship, it is necessary to represent it in a separate diagram, the scheme tree.

Scheme trees represent the hierarchical relationships between object sets. The hierarchical relationships can be modeled directly and n-ary relationships (where $n > 2$) are modeled using more than one scheme tree. Information about the degree of the relationship is lost.

Participation constraints cannot be represented in the scheme tree. However, the representation of participation constraints on the binary relationships is very comprehensive in the CM hypergraph but the meaning of the participation constraints is ambiguous when representing n-ary (n>2) relationships. As there is no distinction between attributes and object classes, the interpretation of "optional" is ambiguous in CM hypergraphs. For example, what is the meaning of "o" on the edge between *course* and *title*? An "o" near *course* represents that a *course* has an optional *title.* Does it make sense to have an "o" near *title,* for a *title* to have an optional *course?* It is worse if there is an "o" in a

ternary relationship, such as an "o" near course, on the edge between *course* and *student.* How do we represent that a *student* is taking a *course* but does not have a *grade* yet?

It is not possible to represent any form of ordering on object sets, for example it is not possible to represent that there is an ordering on *students* within a *course.*

2.5 EER and XGrammar

A language and diagram for modeling XML schemas was defined in [Mani et al., 2001]. The language, called XGrammar, was designed with the aim of capturing the most important features of the proposed XML schema languages. The diagram called the Extended Entity Relationship diagram (EER), differs from other EER diagram notations in that it captures all the concepts that can be represented in Entity Relationship (ER) diagrams while also capturing the hierarchical relationship and ordering on element sets.

The hierarchical relationship or element-subelement relationship is represented using a dummy relationship set labeled "has". The ordering on elements is expressed as a solid line between the relationship set and the ordered entity set. The authors use the term "entity sets" when referring to "element sets". Entity sets are represented as rectangles and relationship sets by diamonds on the edges.

Example 2.7 *Consider the EER diagram in Figure 2.8(a) with entity sets Department, Course and Student. Department has a key attribute called name. Course has a key attribute code, and a single valued attribute title. Student has a key attribute stuNo, single valued attributes stuName and address, and a multi-valued attribute hobby. There are two relationship sets with the label "has", representing a hierarchical relationship between entity sets Department and Course, and another between entity sets Course and Student. The latter has an attribute grade. A department has one or more courses, and a course belongs to only one department. A course has zero or more students and a student belongs to one or more courses.*

Ordering of entities is represented by a bold line in an EER diagram. The requirement that students taking a course must occur in a particular order is represented in Figure 2.9.

The language XGrammar is able to express the hierarchical relationship between entity sets, distinguish attributes from elements, represent participation constraints on the children elements, and represent references. An XGrammar definition of the schema in Example 2.7 is described in Example 2.8. The XGrammar language describes the entity sets and the constraints imposed on them as a 5-tuple {N,T,S,E,A}, where:

1 N is a set of non-terminal symbols, which represent the entity sets.

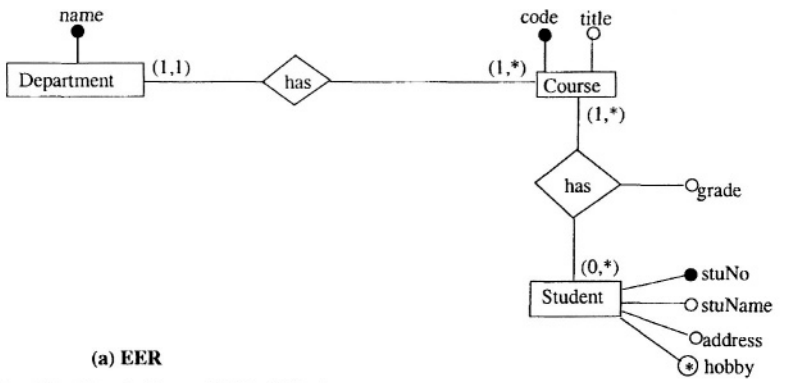

Figure 2.8. An EER diagram and XGrammar definition for Examples 2.7 and 2.8

2 *T* is a set of terminal symbols, which represent instances of the entity sets and attributes.

3 *S* is the non terminal symbol representing the document root.

4 *E* is a set of production rules describing the relationship between the entity sets.

5 *A* is a set of production rules describing attributes.

The production rules in *E* and *A* express the constraints of interest. The authors use the notation ϵ, ~>, and @ to express an empty subelement, a reference, and an attribute respectively.

Example 2.8 *Consider the XGrammar definition in Figure 2.8(b). The set N contains the names of the entity sets, Department, Course and Student, as well as an entity set Has. The relationship set "has" between entity sets Course and Student is modeled as an entity set in the XGrammar definition because it has an attribute, grade.*

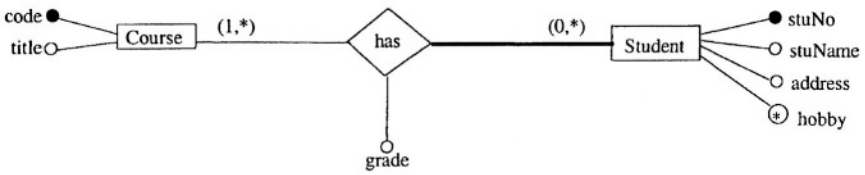

Figure 2.9. An EER diagram and XGrammar definition representing ordering on student within course

Just as in relational data modeling where many-to-many relationships with an attribute are captured in a separate relation, XGrammar models many-to-many relationships with an attribute as a separate entity set. The other "has" relationship set between Department and Course is a one-to-many relationship and is captured in the nesting of Course within Department. The set T contains the entities and attributes. S contains the entity set that is the root of the tree.

The set E specifies the relationship sets on the entity sets. The first rule in E specifies that entity set Department has one or more Courses. Recall the Department represents the entity set while department represents an entity or instance of Department. The second rule specifies that and entity belonging to the entity set Course has zero or more entities of the entity set Has as subelements. The third and fourth rules specify that the entity sets Has and Student have no children. This is represented by the ϵ.

As mentioned above, we have included the entity set Has in the XGrammar definition to deal with the relationship attribute grade. The constraints on attributes are described in set A. An @ denotes an attribute. Entity set Department has an attribute name which is of type string. Entity set Course has two attributes code, and title which is optional. Entity set Has has attributes studentRef which is a reference to Student denoted by ~>, and grade which is optional. Entity set Student has attributes stuNo, stuName, address which is optional, and hobby which is a multi-valued attribute.

How well does the EER and XGrammar support the requirements of a data model for designing a schema for semistructured data? The EER diagram and XGrammar serve different purposes and can in turn represent different concepts. It is not possible to represent an instance of an XML document using EER or XGrammar. In an EER diagram it is not possible to represent which entity set is the root of the tree. There is a problem with representing the hierarchical structure of a semistructured schema in the EER diagram. The relationship set "has" is used to express the hierarchical structure, but this relationship set has no direction so it is unclear which entity set is the element and which is the subelement in the relationship set. So the relationship set "has"

cannot represent the hierarchical structure directly. As shown in Figure 2.8(a) there can be more than one "has" relationship types in an EER diagram.

The hierarchical structure can be represented in XGrammar, but like the DTD, XGrammar represents the hierarchical structure as a binary relationship only. It is possible to represent n-ary as well as binary relationships in the EER diagram but because n-ary relationships were not considered in [Mani et al., 2001], the authors have not considered any implications of these relationships in the algorithms that they specify. It is possible to represent the participation constraints of both child and parent entity sets in the EER diagram, but not in XGrammar. It is possible to show that an attribute is a key or identifying attribute in the EER diagram but it is not possible to show whether other attributes are mandatory or optional. One way to overcome this problem is to represent attributes as entity sets but this could lead to many more entity sets than are really needed. It is not possible to represent identifying attributes in XGrammar.

Like in ER diagrams, it is possible to represent attributes of relationship sets in the EER diagram and there is also an extension to represent ordering on elements. However, there is no distinction between attributes of entity sets and attributes of relationship sets in XGrammar. Also, it is not possible to represent ordering directly in XGrammar. It is not possible to represent ordering on attributes in EER diagrams or XGrammar, except to represent ordered attributes as elements. A distinction is made between attributes and elements in both EER diagrams and XGrammar. The concept of attribute here is the same as that in ER diagrams which differs from the concept in XML documents, so some of the attributes in the EER diagram might be modeled as elements in an XML document.

From the examples and discussion, we can see that EER diagrams are better for conceptual modeling while the XGrammar language is more appropriate for specifying the "implementation" details of the schema. For example in Figure 2.8, information such as there is a many-to-many relationship set between entity sets *Course* and *Student* is lost in the XGrammar model. The fact that *grade* is an attribute of the relationship set between entity sets *Course* and *Student* is also lost. These concepts are modeled using an entity set *Has* and a reference, but the reason for modeling these concepts in this way are not recorded in the XGrammar model.

2.6 AL-DTD and XML Tree

Arenas and Libkin describe a data model that they later use to define a normal form called XNF [Arenas and Libkin, 2004]. In the data model, they define languages for describing an XML Tree and a DTD. In this section we will refer to the DTD defined by Arenas and Libkin as the AL-DTD to avoid confusion.

V = {v0, v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17,v18,v19}

lab(v0) = department	ele(v0) = [v1, v12]	att(v0, @name) = "CS"
lab(v1) = course	ele(v1) = [v2, v3, v7]	att(v1, @code) = "CS1102"
lab(v2) = title	ele(v2) = "Data Structure"	att(v3, @stuNo) = "stu123"
lab(v3) = student	ele(v3) = [v4, v5, v6]	att(v7, @stuNo) = "stu125"
lab(v4) = stuName	ele(v4) = "Zheng Zhang"	att(v12, @code) = "CS2104"
lab(v5) = address	ele(v5) = "hostel 01–03–A"	att(v13, @stuNo) = "stu123"
lab(v6) = grade	ele(v6) = "A"	att(v16, @stuNo) = "stu125"
lab(v7) = student	ele(v7) = [v8, v9, v10, v11]	
lab(v8) = stuName	ele(v8) = "Liang Chen"	
lab(v9) = hobby	ele(v9) = "hockey"	
lab(v10) = hobby	ele(v10) = "reading"	
lab(v11) = grade	ele(v11) = "B"	
lab(v12) = course	ele(v12) = [v13, v16]	
lab(v13) = student	ele(v13) = [v14, v15]	
lab(v14) = stuName	ele(v14) = "Zheng Zhang"	
lab(v15) = address	ele(v15) = "hostel 01–03–A"	
lab(v16) = student	ele(v16) = [v17, v18, v19]	
lab(v17) = stuName	ele(v17) = "Liang Chen"	
lab(v18) = hobby	ele(v18) = "hockey"	
lab(v19) = hobby	ele(v19) = "reading"	

Figure 2.10. A textual representation of the XML Tree in Figure 2.11

An XML Tree is defined precisely in a textual description which can be shown diagrammatically as a tree. The internal nodes are labeled with identifiers, and the leaf nodes are labeled with the value of an attribute or element set.

The textual description is represented as T = (V, lab, ele, att, root) where:

1 V is the set of node identifiers,

2 lab is a mapping from the node identifiers to the names of the element sets and attributes,

3 ele is a mapping from the node identifiers to a list of node identitiers or a string,

4 att is a mapping from the node identifier and attribute name to the value of the attribute.

Example 2.9 *Consider the XML Tree in text format in Figure 2.10 and the diagrammatic representation in Figure 2.11.*

In the diagrammatic representation, identifiers are assigned to the internal nodes of the tree. These identifiers are in turn used in the textual representation

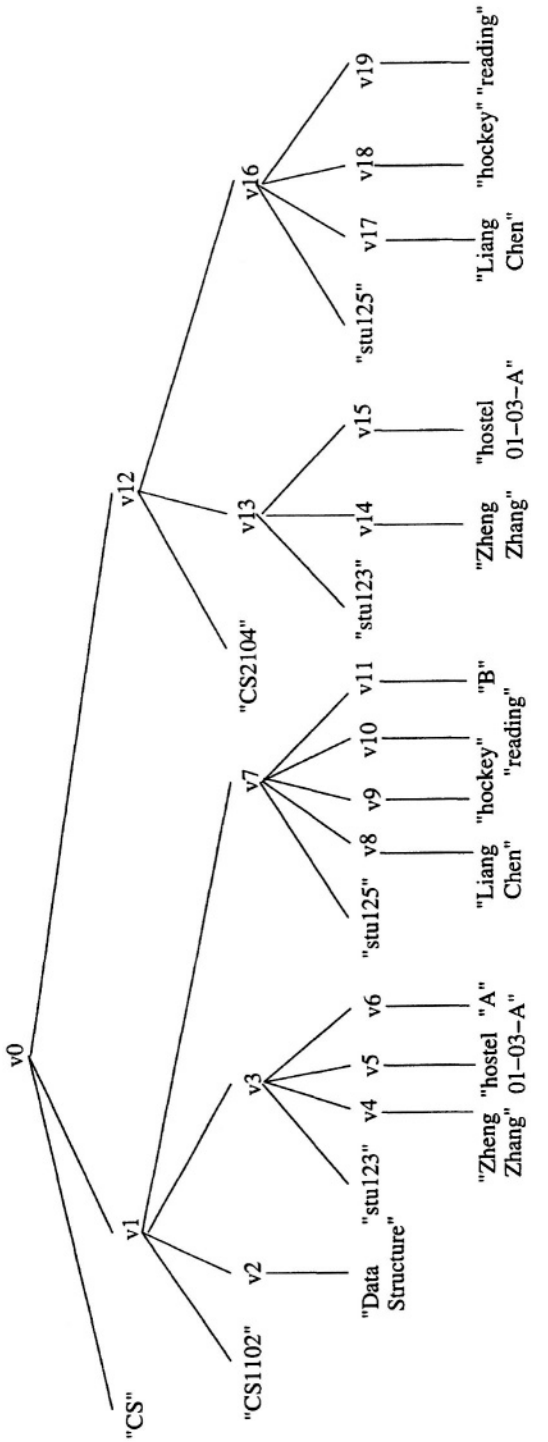

Figure 2.11. A diagram of the XML Tree in Figure 2.10

of the document instance. V is the set of vertices, or node identifiers. The labels in the mapping lab correspond to the values of the tags in the original XML document. These labels include department, course, title, etc. lab provides the mapping from node identifiers to labels.

The mapping ele maps from a node identifier to either a string value (e.g. $ele(v2)$ *= "Data Structures") indicating that the node* $(v2)$ *is a leaf node or a list of node identifiers (e.g.* $ele(v0) = [v1, v12]$*) indicating that the node* (vo) *is a non-leaf node with children* $(v1$ *and* $v12$ *in this case).*

The mapping att maps the node id and attribute name to the value of the attribute. For example, the mapping $att(v0, @name) =$ *"CS" shows that node with id* $v0$ *has an attribute called name with value "CS".*

One of the deficiencies of the textual representation is that it is difficult to visualize the data and their relationships. The node identifiers in the diagrammatic representation and in *V* in the textual representation are introduced and bear no relationship to the original XML document. The diagrammatic representation alone also has a number of deficiencies. In particular, the labels of nodes and attribute names are not shown; the relationships captured are binary relationships; and it is not possible to distinguish between attributes of elements and attributes of relationships, e.g. *stuName* and *grade* are represented in the same way. Notice that the ordering of the children is significant.

The schema of the XML Tree, represented in an AL-DTD, can be represented precisely in a language that captures similar semantics to the document type definition (DTD) language [Bray et al., 2000].

The AL-DTD is represented as D = (E, A, P, R, r) where:

1 E is a set of element sets,

2 A is a set of attributes,

3 P is a mapping from element sets to the children of the element sets, indicating the participation constraints on children,

4 R is a mapping from element sets to attributes, indicating which element set the attribute belongs to,

5 r is the name of the root element set.

Example 2.10 *Consider the AL-DTD in Figure 2.12. The set E contains all the element sets, such as department, course, title, etc. and the set A contains all the attributes, such as name, code, stuNo. The set* r *contains the root element set, department. P maps from the element sets to the children of the element sets, for example course has children title and* $studentRef(s)$*. Department must have at least one course, represented as course, course*.*

E ={department, course, title, student, stuName, address, hobby, grade, studentRef}
A = {@name, @code, @stuNo}
r = department
P(department) = course, course*,student*
P(course) = title, studentRef*
P(title) = S
P(student) = stuName, address, hobby*
P(studentRef) = grade
P(name) = S
P(address) = S
P(hobby) = S
P(grade) = S
R(department) = {@name}
R(course) = {@code}
R(student) = {@stuNo}
R(studentRef) = {@stuNo}

Figure 2.12. An AL-DTD schema for the XML Tree in Figures 2.10 and 2.11

An element set that has a string value is mapped to S, which corresponds to PCDATA in the DTD. R which is the mapping from element sets to attributes shows that element set department has attribute name etc.

How well does the AL-DTD and XML Tree support the requirements of a data model for designing a schema for semistructured data? This data model is able to succinctly and precisely represent both the instance and schemas of XML documents. However, since it is based on XML documents, it has the same shortcomings as XML. While it handles hierarchical relationships nicely, it is not possible to represent many-to-many and many-to-one relationships. It is not possible to distinguish between binary and n-ary ($n > 2$) relationships and there is also no distinction between attributes of element sets and attributes of relationships. The AL-DTD does not show identifying attributes, and because the schema is modeled as a tree it is not possible to model IDREF or IDREFS from the DTD language directly. The AL-DTD is less expressive than the DTD language.

2.7 ORA-SS

The ORA-SS data model [Dobbie et al., 2000] has four basic concepts: object classes, relationship types, attributes and references, and consists of four diagrams: the schema diagram, the instance diagram, the functional dependency diagram and the inheritance diagram. Notice that "element sets" are called "object classes" in the ORA-SS data model, and "relationship sets" are called "relationship types".

The ORA-SS instance diagram is like a DOM tree, in that it captures the instance of a document. However, it is unlike the DOM tree in that it is possible to distinguish between attributes and object classes. The ORA-SS schema diagram, which captures the schema information, is like the CM hypergraph, except that it is semantically richer. The functional dependency diagram captures functional dependencies that cannot be expressed in the participation con-

straints in the schema diagram, such as those of n-ary relationships, and the inheritance diagram captures ISA relationships.

Consider the ORA-SS instance diagram in Figure 2.13. An ORA-SS instance diagram has two kinds of nodes, internal and leaf nodes. Internal or non-leaf nodes are represented as labeled rectangles, and leaf nodes are labeled circles that have a value. The rectangles represent objects and the circles represent attributes and their values. Attributes in the ORA-SS sense are like attributes in an ER diagram which is not the same as attributes in XML documents. Attributes in ORA-SS may be represented as elements or attributes in an XML document. ORA-SS instance diagrams can be used when an XML document exists but the schema of the document is unknown. Some schema information can be extracted from the instance diagram while other semantic information and constraints must be sought from a domain expert. See Chapter 4 for details.

Consider the ORA-SS schema diagram in Figure 2.14. An object class is represented as a labeled rectangle. A relationship type between object classes in an ORA-SS schema diagram can be described by *name (object class list), n, p, c,* where *name* denotes the name of the relationship type, *object class list* is the list of objects participating in the relationship type, *n* is an integer indicating the degree of the relationship type (n=2 indicates binary, n=3 indicates ternary, etc.), *p* is the participation constraint (the cardinality of the mapping between object classes) of the parent object class in the relationship type, and *c* is the participation constraint of the child object class. The *name* is optional. The *object class list* is included only if the object classes participating in the relationship type are separated by some other object class(es) not relevant to the relationship type in the path which includes all participating object classes of the relationship type. The participation constraints are defined using the *min:max* notation, also used in the EER diagram. The symbols ?, *, + are shorthand notations and have the same meaning as they have in DTDs. An edge between two object classes can have more than one such relationship type label to indicate the different relationship types the object classes participate in.

Attributes of object class or relationship type are denoted by labeled circles. Some object classes may have identifiers, which are denoted as filled circles. An attribute can be mandatory or optional, single-valued or multivalued. All attributes are assumed to be mandatory and single-valued, unless the circle contain an ?, which shows that the are single valued and optional, or a + which shows that they are multivalued and required, or an * which shows they are optional multivalued attributes. Attributes of an object class can be distinguished from attributes of a relationship type. The former has no label on its incoming edge while the latter has the name of the relationship type to which it belongs on its incoming edge.

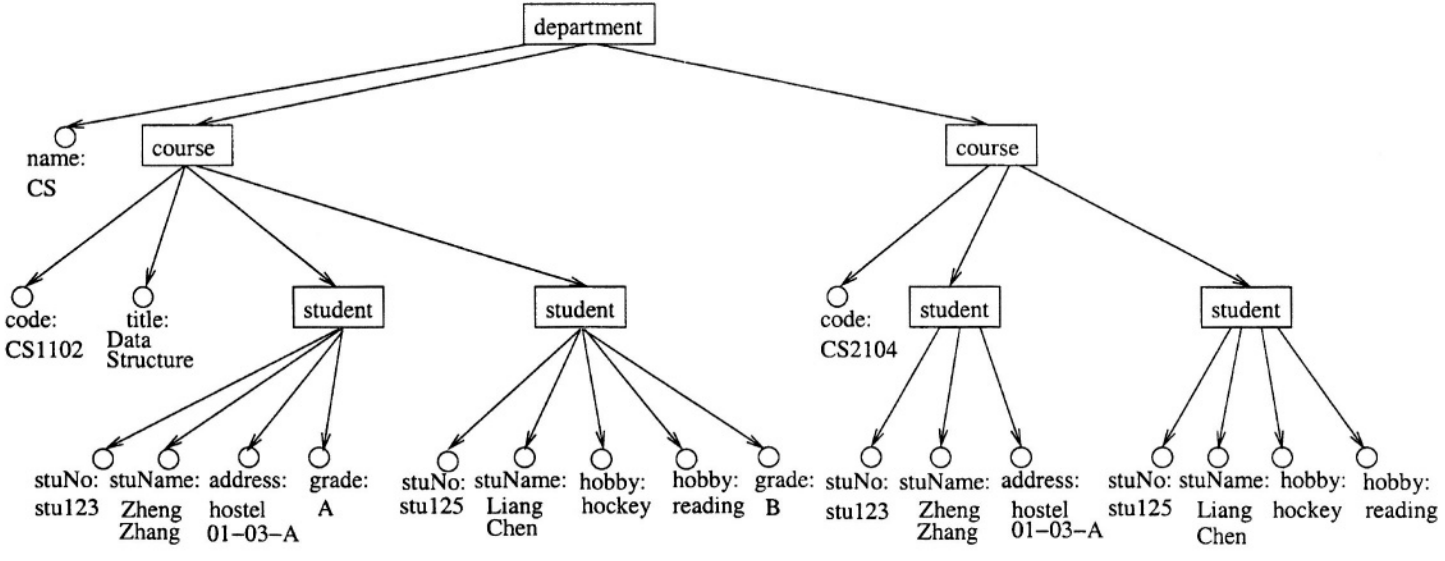

Figure 2.13. An ORA-SS Instance Diagram for the document in Figure 2.1

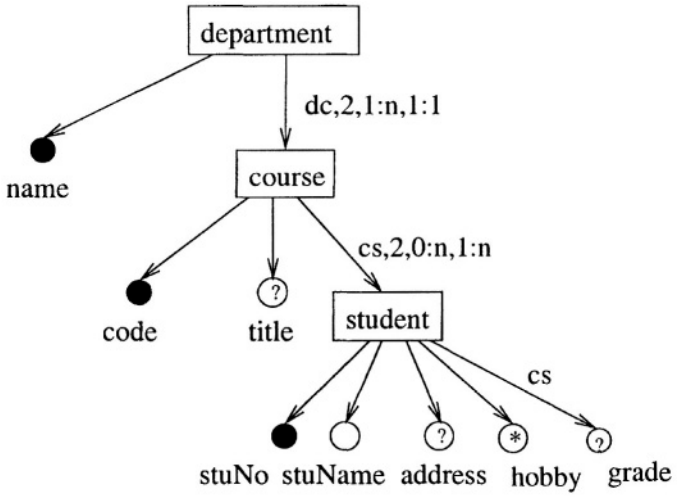

Figure 2.14. An ORA-SS schema diagram for the document in Figure 2.1

Example 2.11 *Consider the ORA-SS instance diagram in Figure 2.13. The labeled rectangles represent objects, for example, there are objects department, course and student. The labeled circles represent attributes, for example there is an attribute code with value "CS1102", and the attribute title with value "Data Structure".*

Figure 2.14 show the ORA-SS schema diagram. The rectangles labeled department, course and student are examples of object classes. Attributes name, code and stuNo are the identifiers of the object class department, course and student respectively. The meaning of identifier in the ORA-SS is the same as the identifier of an entity type in the ER sense, that is, the attribute has a unique value in the real world. For example, every student has a unique stuNo. Each of the attributes title, address, hobby and grade are optional. Attribute hobby is multivalued, and attribute stuName is required.

There are two relationship types, called dc and cs. Relationship type dc represents a binary relationship between object classes department and course, and cs represents another binary relationship between course and student. A department can have one or more courses, and a course belongs to one and only one department. A course can have zero or more students, and a student can take 1 or more courses. The label cs on the edge between student and grade indicates that grade is a single valued attribute of the relationship type cs. That is, the attribute grade is the attribute of a student in a course. From

these constraints, we can derive that

$$course,\ student \rightarrow grade.$$

Some of the information in the label is redundant so it is not necessary to include all the fields. For example the label for relationship type dc can be shortened to *2,+,1:1,* since we do not need to use the name of the relationship type elsewhere and it is obvious that the relationship type is between object classes *department* and *course.*

How well does the ORA-SS data model support the requirements of a data model for designing a schema for semistructured data? The hierarchical structure of the object classes is clearly shown in the ORA-SS schema diagram, along with participation constraints on parent and children object classes. A distinction is made between attributes of object classes and attributes of relationship types. An attribute that belongs to a relationship type has the name of the relationship type on its incoming edge where an attribute that belongs to an object class has no label on its incoming edge. Compare for example the attributes *address* and *grade.* Attribute *address* is an attribute of *student* where attribute grade is an attribute of the relationship type *cs* where the participating object classes are *course* and *student.* In Figure 2.14 we depict only binary relationships but it is also possible to represent n-ary relationship types ($n > 2$) in ORA-SS schema diagrams.

Example 2.12 *The ORA-SS schema diagram in Figure 2.15 shows a relationship type between student and course called sc. Attribute grade is an attribute of relationship type sc. The attribute grade models the grade that a student gains in a course. There is a binary relationship type between course and tutor called ct, with an attribute hours. The attribute hours models the hours a tutor spends on a course per week. Finally there is a relationship type called sct among object classes student, course and tutor. It is a ternary relationship. Attribute feedback, which belongs to relationship type sct, models the feedback a tutor gets on a course from a student.*

It is also possible to depict orderings on attributes or elements. For example if students are to be ordered within a course, this is depicted by a < in the label, and if hobbies are ordered by priority this is depicted by a < on the incoming edge as shown in Figure 2.16.

A more detailed description of the ORA-SS data model in given in Chapter 3.

2.8 Discussion

In this chapter, we have listed the features that are required in XML data models to support the design of schemas for XML documents. We have de-

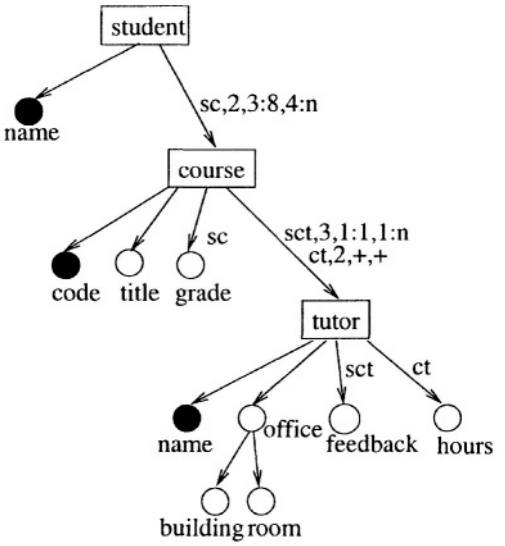

Figure 2.15. An ORA-SS schema diagram showing binary and ternary relationships

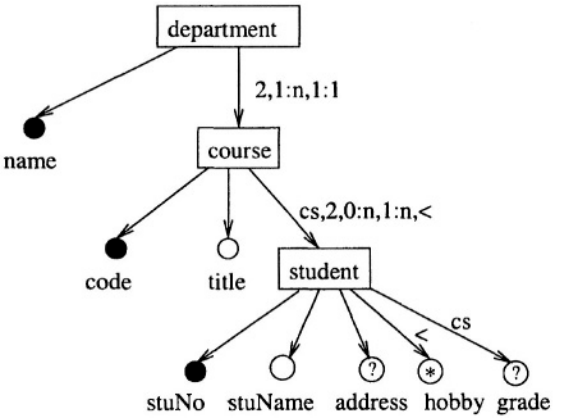

Figure 2.16. An ORA-SS schema diagram showing ordering of students and hobbies

scribed the major data models that have been proposed for modeling schemas for semistructured documents. Table 2.2 summarizes the findings of this chapter. In this table, a $\surd$ indicates the data model supports the feature, a $\times$ indicates that data model does not support the feature, while a $\surd\times$ indicates that the feature is partially supported or the feature is not directly supported but it is possible to model it.

	DTD	DOM	OEM	S3-graph	CM	EER	XMLTree	ORA-SS
instance	×	√	√	×	×	×	√	√
schema	√	×	√	√	√	√	√	√
identifying attribute	√×	×	×	×	×	√×	×	√
hierarchical relationship	√	√	√	√	√	√×	√	√
nary relationship	×	×	×	×	√	√	×	√
participation constraints	√×	×	×	√×	√×	√	√×	√
references	√×	×	×	√×	×	√	√×	√
attributes and elements	√×	√	×	√	×	√	√	√
relationship attributes	×	×	×	×	√×	√	×	√
ordering	√×	×	×	×	×	√	√×	√

Table 2.2. Features supported in XML Data Models

A DTD does not model the instance of the data, and the only relationships that can be modeled using the DTD are binary relationships in the hierarchical structure or references. It is possible to indicate the participation of the children in hierarchical relationships and the participation of the referenced elements. No distinction is made between attributes of elements and attributes of elements, and there is always an ordering on subelements since they are represented as a sequence.

A DOM represents an instance of an XML document, so it is not possible to represent information about the schema other than the hierarchical structure. A DOM does distinguish between attributes and elements. The OEM again represents an instance of an XML document, but unlike the DOM it does not distinguish between attributes and elements. With DataGuides, the hierarchy of object sets is represented explicitly. An inadequacy of the DataGuide is its inability to express the degree of n-ary relationships for the hierarchical semistructured data, which introduces ambiguous data representations.

The S3-graph does not model the instance of an XML document, but captures the hierarchical structure of semistructured data. It does not provide for the specification of the identifying attribute of an entity, and does not distinguish between attributes of objects and relationships. The S3-graph is not able

to depict the degree of relationships, and cannot capture relationships beyond the binary one-to-one and one-to-many relationships.

The CM hypergraph and scheme tree data model has no way to represent a data instance, does not provide a way to model references, does not distinguish between objects and attributes, has no way of representing ordering, and the conceptual information and nesting relationships are in two separate diagrams. Participation constraints and the distinction between attributes of relationships and attributes of elements can be modeled in CM hypergraphs.

Using the EER and the XGrammar data model it is not possible to represent a data instance, and the representation of hierarchy in the EER diagram is ambiguous without other information. While it is possible to represent many of the other features, it is sometimes necessary to have both the EER representation and the XGrammar representation to gain a clear understanding of the exact semantics. For example, if a relationship is modeled using references in the XGrammar representation then some participation constraint information is lost. Also, XGrammar does not distinguish between attributes of object classes and attributes of relationship types, and XGrammar is unable to represent the degree of relationship types. Because it is not possible to model all the constraints in either EER or XGrammar, both representations are required to model the semantics needed to effectively manage semistructured data.

The data model presented in [Arenas and Libkin, 2004] describes a language for specifying an instance, (i.e. the XML Tree), a diagram for representing the instance and a language for specifying the schema (i.e. the AL-DTD). The specification of the instance, represents the hierarchical structure of the XML instance, the values of elements and attributes, and the ordering on elements. The language for specifying the schema captures most of the of the DTD language, except ID and IDREF(S). Consequently it is not possible to specify an n-ary relationship where $n > 2$, it is not possible to specify the participation constraint of a parent element in a relationship, and it is not possible to distinguish between attributes of element sets and attributes of relationship types.

In summary, the major advantages of ORA-SS over existing semistructured data models for designing schemas for semistructured data are its ability to represent the data instance, distinguish between attributes and object classes, differentiate between attributes of object classes and attributes of relationship types, and to express the degree of relationship types and the participation constraints on the object classes in the relationship types. Such expressed information is important, even crucial for designing "good" semistructured databases, and defining meaningful semistructured views.

http://www.springer.com/978-0-387-23568-4

Semistructured Database Design
Ling, T.W.; Dobbie, G.
2005, XVI, 178 p.,
ISBN: 978-0-387-23568-4