ORA-SS: An Object-Relationship-Attribute Model for Semi-Structured Data

Gillian Dobbie Wu Xiaoying Tok Wang Ling
Mong Li Lee
Department of Computer Science, National University of Singapore, Singapore
{dobbie,wuxiaoy1,lingtw,leeml}@comp.nus.edu.sg

Abstract

Semi-structured data is becoming increasingly important with the introduction of XML and related languages and technologies. The recent shift from DTDs (document type definitions) to XML-Schema for XML data highlights the importance of a schema definition for semi-structured data applications. At the same time, there is a move to extend semi-structured data models to express richer semantics. In this paper we propose a semantically rich data model for semi-structured data, ORA-SS (Object-Relationship-Attribute model for Semi-Structured data). ORA-SS not only reflects the nested structure of semi-structured data, but it also distinguishes between objects, relationships and attributes. It is possible to specify the degree of n-ary relationships and indicate if an attribute is an attribute of a relationship or an attribute of an object. Such information is lacking in existing semi-structured data models, and is essential information for designing an efficient and non-redundant storage organization for semi-structured data.

1 Introduction

There has been increased interest in semi-structured data recently with the introduction of XML and related languages and technologies. Semi-structured data ranges from completely structured data to completely unstructured data, and usually falls somewhere between these extremes. That is, usually part of the data has structure and part has no consistent structure. Initially it was claimed that semi-structured data is self-describing, and it isn't important to define a schema for the data. The move from DTD [3] (document type definition), a simple schema definition language, to XML-Schema [31], a more expressive schema definition language highlights the importance of a schema definition for semi-structured data applications. Similarly there is a move to extend semi-structured data models to express richer semantics. The data models that have been proposed specifically for semi-structured data, e.g. those described in [5, 8, 14, 23] are defined for at least one of the two following purposes: for finding a common schema for two or more heterogeneous data sources or for extracting a schema from a semi-structured document.

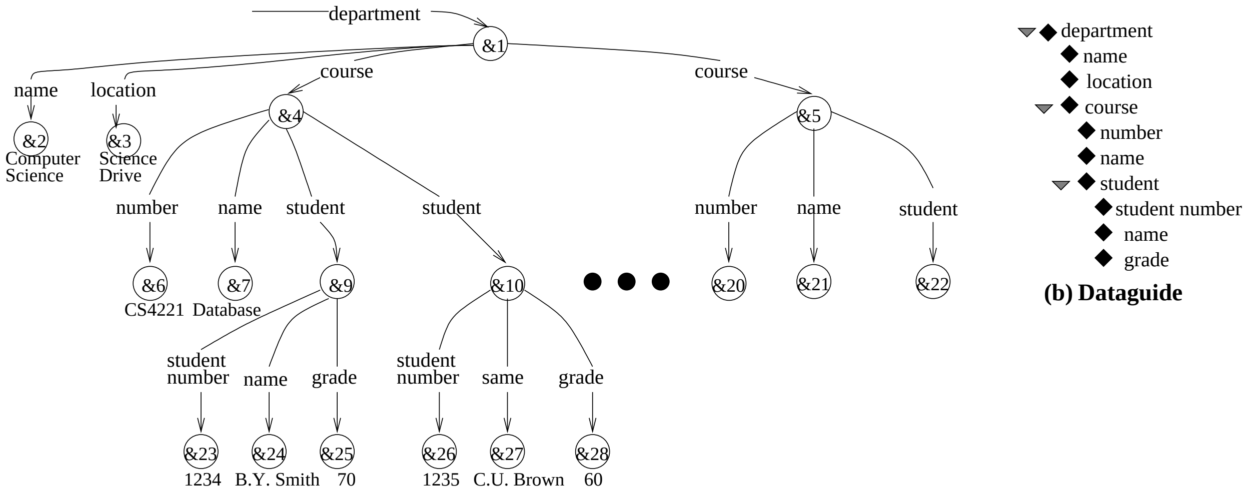
In this paper we describe a richer data model for semi-structured data, ORA-SS (Object-Relationship-Attribute model for Semi-Structured data). The richer semantics of ORA-SS enables us to capture more of the real world semantics, and use them in storage organization and view design. Semi-structured data is usually stored in flat files or in repositories, like relational database repositories [27], object-oriented repositories [20] or in specialized semi-structured systems such as Strudel [11], Lore [21], Lotus Notes [9] and Tamino [29]. Like in traditional database systems, efficient storage and access, and minimizing update anomalies is important and this cannot be done without knowledge of the semantics of the data. One of the key advantages of semi-structured data, and XML in particular is that the data is stored with no presentation details, and style sheets are used to describe how the data is to be presented. There can be many different style sheets for the same data. Style sheets can be likened to view definitions in traditional database applications. In order to define meaningful and easily updated presentations (views) of the data, the designer must know the semantics of the data.

The ORA-SS data model distinguishes between objects, relationships and attributes. The relationships between objects are expressed explicitly. The main contributions of this paper are expressing the degree of an n-ary relationship, and distinguishing between attributes of relationships and attributes of objects. Knowing the degree of an n-ary relationship leads to more efficient storage and access to the data. Distinguishing between relationship attributes and object attributes enables us to describe which attributes and relationships are reachable from an object that references another object. For example, consider the scenario where `student`s are nested within `course`s, and each `student` has a `name`, `address` and `grade` attribute. (The `grade` attribute is the grade of a student in a course, so it is an attribute of the relationship between `course` and `student`.) Bookshops may be given access to student objects to get their name and mailing address. This can be achieved using a reference between the `bookshop` object and `student` object. Being attributes of the object `student`, attributes `name` and `address` have meaning when they are accessed from `bookshop`, but because `grade` is an attribute of the relationship between `course` and `student`, it has no meaning when it is accessed from `bookshop`. In most semi-structured data models, an attribute of a relationship is treated no differently than an attribute of an object, and so it is impossible to distinguish which attributes are reachable from a referencing object, and which are not.

The rest of the paper is organized as follows. Section 2 contains a motivating example. Section 3 describes the ORA-SS data model. In section 4 we outline how DTDs are mapped to ORA-SS schema diagrams and how ORA-SS schema diagrams are mapped to relational databases. Section 5 compares the ORA-SS data model with other data models that have been proposed for semi-structured data. Section 6 concludes the paper, highlighting other areas in which ORA-SS will be used and some future directions of research.

2 Motivation

One way to display the schema of semi-structured data is using dataguides [14]. Consider the instance in Figure 1(a) represented using OEM [21], and the associated dataguide in Figure 1(b). The dataguide is derived from the instance of the data. A dataguide adequately describes the nested structure of the data, however other semantic information cannot be modeled using a dataguide. In particular, it is not possible to represent the degree of n-ary relationships, or whether an attribute is an attribute of an object or the attribute of a relationship. This kind of information is essential for designing data repositories for semi-structured data that are easy to maintain.



(b) Dataguide

(a) OEM Diagram

Figure 1: Sample instance demonstrating OEM and Dataguides

In Figures 2 and 3 we represent the instance described in Figure 1 using an ORA-SS instance diagram and an ORA-SS schema diagram, respectively. Figure 2 is an ORA-SS instance diagram showing the instance, highlighting the difference between objects and attributes. Objects are represented as labeled rectangles, attributes as labeled circles.

Figure 3 shows the schema, distinguishing between objects, relationships and attributes, highlighting the degree of n-ary relationships, the participation of objects in relationships and whether an attribute is a relationship attribute or an object attribute. The primary key of *department* is *name* (indicated by the filled circle) and *department* has another attribute *location*. The label on the edge between *department* and *course* (2,1:n,1:1) indicates that there is a binary relationship between *department* and *course* (2), there can be many *course*s in each *department* (1:n) and that a *course* belongs to only one *department* (1:1). The primary key of *course* is *number* and *course* has another attribute *name*, that is not necessarily unique. In fact, the default cardinality of *name* is $0:1$ so not every *course* is expected to have a *name*. Every *student* has a *student number* and may have a *name*, where the *student number* is unique. The

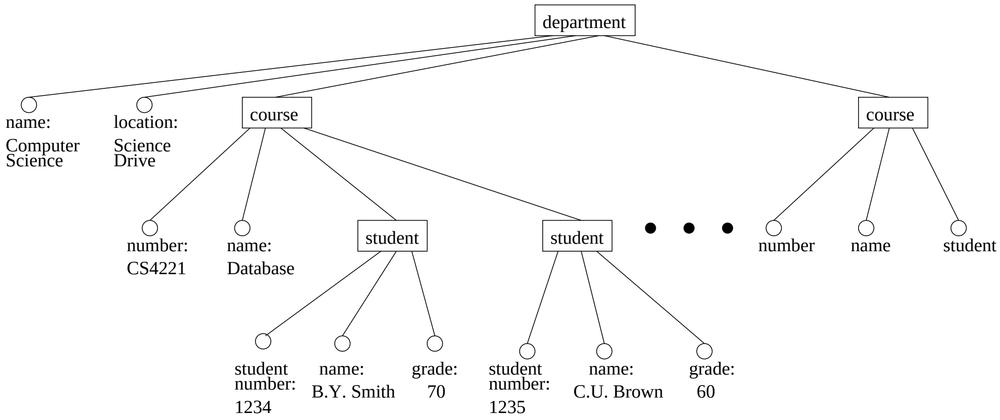


Figure 2: ORA-SS Instance Diagram

binary many-to-many relationship between *course* and *student* has the name *cs*. The label *cs* on the edge between object *student* and attribute *grade* indicates the attribute *grade* belongs to relationship *cs* rather than the object *student*. How does this semantic information help when repositories are being designed? Because *cs* is a many-to-many relationship, we know that if we nest *student* within *course*, the attributes of student will be repeated for every *course* they take. Also, because *cs* is a many-to-many relationship, the relationship attribute *grade* cannot be stored in either *course* or *student* and must be stored in something that represents the relationship between the objects.

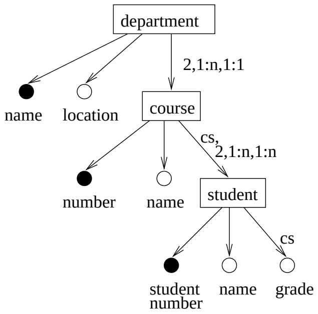


Figure 3: ORA-SS Schema Diagram

It is not a coincidence that ORA-SS diagrams are similar to (but not the same as) entity-relationship (ER) diagrams. The advantage of ORA-SS diagrams being similar to ER diagrams is that people who are familiar with ER diagrams can readily understand the concepts underlying the ORA-SS data model. The advantage of ORA-SS diagrams over ER diagrams for modeling semi-structured data is that the nesting of the objects is reflected directly in the ORA-SS diagrams. There are other concepts that can be modeled in ORA-SS diagrams and not in ER diagrams like the ordering of elements and attributes, and fixed and default attribute values.

3 The Data Model

The ORA-SS data model has three basic concepts: objects, relationships and attributes. Objects are similar to entities in the ER model and classes in the object-oriented data model. They coincide with elements in XML. An object is related to another object through a relationship. Nesting and referencing are the only relationships in the semi-structured data model. The ORA-SS data model represents both these relationships very naturally. Attributes are properties, and may belong to an object or a relationship. This distinction is made in the ORA-SS data model. The ORA-SS data model consists of 4 diagrams: ORA-SS schema diagram, ORA-SS instance diagram, functional dependency diagram, and ORA-SS inheritance diagram. We refine the semi-structured data concepts in the following sections and describe how the following constraints are represented in the ORA-SS data model:

- object
  - attributes of objects
  - ordering on objects
- relationship
  - attributes of relationships
  - degree of n-ary relationships
  - participation of objects in relationships
  - disjunctive relationships
  - recursive relationships
  - symmetric relationships
- attribute
  - key attribute
  - cardinality of attributes
  - composite attributes
  - disjunctive attributes
  - attributes with unknown structure
  - ordering on attributes
  - fixed and default values of attributes
- semi-structured data instance
- functional dependencies and other constraints
- inheritance hierarchy.

3.1 Objects

An object is like a set of entities in the real world, an entity type in an ER diagram, a class in an object-oriented diagram or an element in the semi-structured data model. Traditionally, an entity type has a name and is characterized by the set of attributes that belong to each entity. Because semi-structured data can be less structured than traditional (structured) data, objects are characterized by a name rather than a set of attributes. In the semi-structured data model, the set of attributes associated with an object are the attributes that instances of that object could have and there is no expectation that every instance of an object has exactly the same set of attributes.

An object is represented as a labeled rectangle. The attributes are represented as labeled circles joined to their object by an edge. Keys are filled circles. We give more detail about attributes and keys in Section 3.3.

Example 3.1 Consider an example where each student can have a number, a first name and a last name, and the key is number. This is represented in Figure 4 by an object *student* with key *number*, and attributes *first name* and *last name*.

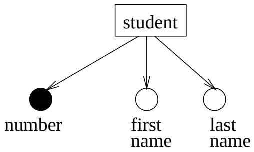


Figure 4: Object *student* with attributes in an ORA-SS schema diagram

3.2 Relationships

Two objects are connected via a relationship. A relationship in the ORA-SS data model represents a nesting relationship. Each relationship has a degree and participation constraints. A relationship of degree 2 (i.e. a binary relationship) relates two objects. One object is the parent and the other the child, and we distinguish between the participation constraint on the parent in the relationship and the participation constraint on the child in the relationship. A relationship of degree 3 (i.e. a ternary relationship) is a relationship between three objects. In a tertiary relationship, there is a binary relationship between two objects, and a relationship between this binary relationship and the other object. The parent, in this case, is the binary relationship, and the child is the other object.

A relationship is represented in an ORA-SS schema diagram by a labeled diamond. The diamond is optional and is assumed on any edge between two objects. The label, $name,\ n,\ p,\ c$, contains a relationship $name$, an integer n indicating the degree of the relationship ($n = 2$ indicates binary, $n = 3$ indicates ternary, etc.), the participation constraint p on the parent of the relationship, and the participation constraint c

on the child. By defining participation constraints with min:max notation, we are also able to represent numerical constraints. The usual shorthand can also be used to represent the participation constraints, ?(0:1), *(0:n), +(1:n). All fields in the label are optional. There is no default value for name. The default value for degree is 2. The default value for the parent participation constraint is $0 : n$ and the default value for the child participation constraint is $1 : m$.

Example 3.2 Figure 5(a) shows a binary relationship between *project* and *member* and a binary relationship between *member* and *publication*. The relationship between *project* and *member* is annotated with $2, +, +$, which represents a many-to-many relationship between *project* and *member*, and a total participation constraint on both objects. Figure 5(b) shows an instance of this schema with a relationship between *project*s and *member*s, and another between *member*s and *publication*s, but no relationship between *project*s and *publication*s. From this diagram, we can deduce that member $m1$ has publications $pub1$, $pub2$ and $pub3$, but we don't know which project the publications are associated with. A dataguide for this schema is shown in Figure 5(c). Notice that there is no relationship between project and publication in the ORA-SS schema diagram in Figure 5(a) and if the data is nested as is suggested in Figure 5(a), then all the publications for each member will be repeated for every project the member works on.

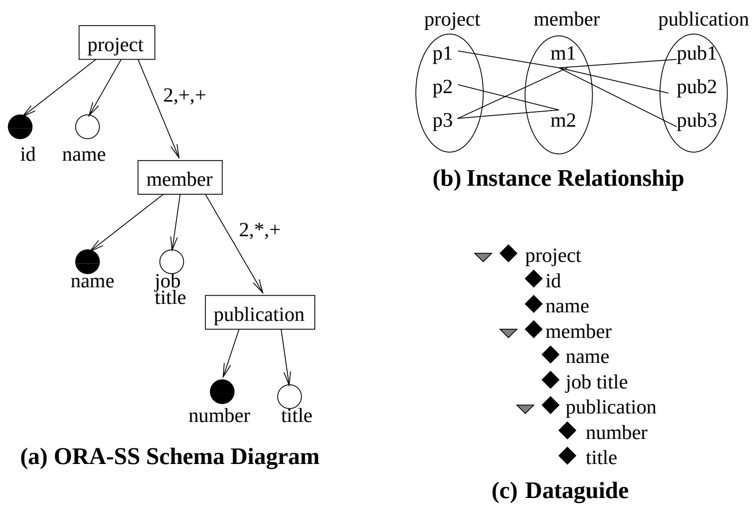


Figure 5: Representing binary relationships

In contrast, Figure 6(a) shows a ternary relationship between *project*, *member* and *publication*. There is a binary relationship (named *pm*) between *project* and *member*, and a relationship (named *mp*) between *pm* and *publication*. Figure 6(b) shows an instance of this schema. It shows a relationship between *project* and *member*, and another relationship between the *project* and *member* relationship and *publication*s. From this diagram, we can deduce that publications $pub1$ and $pub2$ are associated with member $m1$ and project $p1$. Note that a dataguide for this schema is the same as that in Figure 5(c) although the constraints on the relationships in the ORA-SS diagrams are quite different. The schema in Figure 6(a) models the relationship between papers written by a particular member while working in a particular project, and if the data is nested as is suggested in Figure 6(a) then only the publications written by a member while working on a project will be nested within that member and project. This distinction between binary and ternary relationships cannot be made in other semi-structured data models.

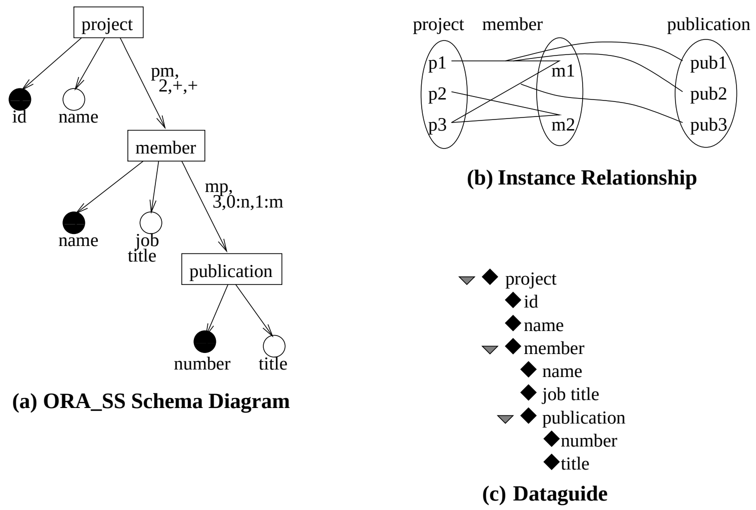


Figure 6: Representing ternary relationships

3.3 Attributes

Attributes represent properties. An attribute can be a property of an object or a property of a relationship. The attributes we have seen until now in this paper are simple attributes of objects. One of the features that distinguishes semi-structured data from structured data is that not all objects and relationships are expected to have the same set of attributes, and because of this the attributes of instances of objects are heterogeneous. An attribute can be a:

- composite attribute, i.e. be composed of other attributes,
- candidate key, i.e. an attribute that has a unique value for each instance of an object,
- composite key, i.e. a set of attributes that have a unique value for each instance of an object,
- primary key, i.e. a chosen candidate key,
- single-valued, i.e. has only one value,
- multi-valued, i.e. can have a set of values,
- required, i.e. must have a value for every instance,
- optional, i.e. may not have a value in some instances,
- fixed value, i.e. the value is the same for every instance and cannot be changed,

- default value, i.e. the value is the default value if no other value is specified for the attribute,
- derived i.e. the value is derived or computed, or
- have unknown structure or an attribute may have a different structure for different instances.

An attribute is represented by a labeled circle. The label consists of *name*, $[F|D\ :\ value]$. The *name* is compulsory, and the rest of the label is optional. The letter F precedes a fixed value, while D precedes a default value. The special attribute name **ANY** denotes an attribute of unknown or heterogeneous structure. An attribute that is the primary key is a filled circle, while other candidate keys are a double circle with the inner circle filled. A composite key is shown by drawing a line across the edges leading to the attributes that form the key. Composite attributes are represented by a labeled circle with edges to the component attributes. A derived attribute is denoted by a dashed circle. An attribute's cardinality is shown inside the attribute circle, using ?(0:1), *(0:n), +(1:n), where the default is 0:1. If we want to be more specific, we can use the min:max notation to represent participation constraints, where min is the minimum cardinality and max is the maximum cardinality of the attribute. An attribute with no label on its incoming edge is an attribute of an object. A relationship attribute has the name of the relationship to which it belongs written on its incoming edge. The attributes shown for an object or relationship in an ORA-SS diagram are the possible objects that the object or relationship may have. An instance of that object or relationship would have a subset of the attributes shown.

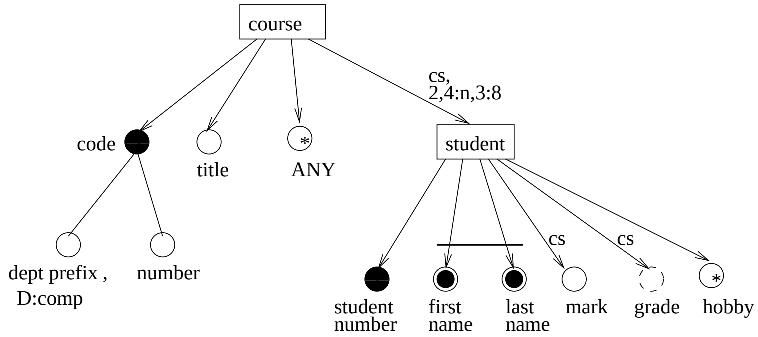


Figure 7: Object with relationships and attributes in an ORA-SS Schema Diagram

Example 3.3 Consider the ORA-SS schema diagram in Figure 7. The object *course* has attributes *code*, *title* and an attribute whose structure is unknown. Attribute *code* is the primary key of *course* and is a composite attribute. It is made up of the attributes *dept prefix* and *number*. The default value of *dept prefix* is *comp*, denoted as $D : comp$. Object *student* has a primary key *student number*, a composite candidate key made of the attributes *first name* and *last name*, a multi-valued attribute *hobby*, and two other single-valued attributes. The attribute *mark* belongs to the relationship, *cs*, between *course* and *student*, i.e. it is the mark for a student in a course. Attribute *grade* is a derived attribute of relationship *cs*, whose value is calculated from the student's mark in the course.

Notice that the key is unique among the instances of an object and the uniqueness is not related to the path you take to reach the instance. Consider for example, the chapters of a book. The chapter number is not a key because it is not unique across all books.

3.4 Ordering

One of the features of semi-structured data is that some kind of ordering is enforced. For example, in XML, ordering is enforced on elements. We distinguish between 3 kinds of ordering in the ORA-SS data model:

- the instances of an object can be ordered,
- the values of an attribute can be ordered, and
- the set of attributes of an object can be ordered.

Consider the scenario where authors of a paper are modeled as objects, and the order in which the authors are listed is important, then this can be modeled in the ORA-SS data model by indicating the ordering on the objects. If, on the other hand, the authors were simply modeled as attributes and the ordering is important, we can model this in ORA-SS by showing an ordering on the attributes. A book is made up of a number of different sections, e.g. the table of contents, the chapters, an index, an appendix, and the order of the sections is important. This is modeled by showing an ordering on the attributes of an object.

In the ORA-SS schema diagram, we use the following notation:

- symbol “<” after a relationship label indicates that instances of the child object are ordered,
- symbol “<” on an edge between and object and an attribute indicates that the values of the attribute are ordered, and
- symbol “<” adjacent to an object indicates that the set of attributes of the object are ordered.

Example 3.4 Consider the ORA-SS schema diagram in Figure 8. The symbol “<” on the edge between object *book* and attribute *author* indicates that the order of the authors is important. The symbol “<” adjacent to object *content* indicates that the order of the set of attributes of this object is important i.e. the structure of *content* is attribute *preface*, followed by attribute *toc*, followed by the object *chapter*. The symbol “<” following the relationship label for the relationship *cc* indicates that the ordering of the instances of object *chapter* is important.

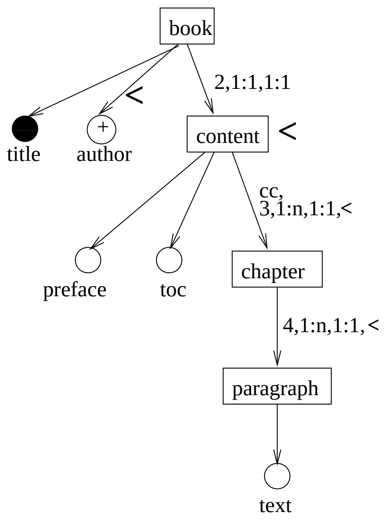


Figure 8: Ordered objects and attributes in an ORA-SS Schema Diagram

3.5 Disjunction

A characteristic of semi-structured data is that attributes and objects are likely to be less homogeneous than in structured data. To allow for this, we provide for two different kinds of disjunction in the ORA-SS data model:

- disjunctive objects, and
- disjunctive attributes.

A disjunctive relationship is used to represent disjunctive objects, such as a student lives in a hostel OR at home, and is represented by a relationship diamond labeled with symbol "|". A disjunctive attribute is represented by a circle labeled with symbol "|", with edges from this circle to the alternatives.

Example 3.5 Consider the schema diagram in Figure 9. A *course* has a *code*, *title*, and a disjunctive attribute *exam venue*. Attribute *exam venue* is a disjunctive single-valued attribute, and can be a *lecture theatre* or a *laboratory*, but not both. It is denoted by a "|" inside a circle. Relationship *sh* is a disjunctive relationship, and is denoted by a "|" inside a diamond. It represents the fact that a *student* can either live in a *hostel* or at *home*. The participation constraints indicate that an instance of this relationship is mandatory, and a *student* must live in either a *hostel* or *home*.

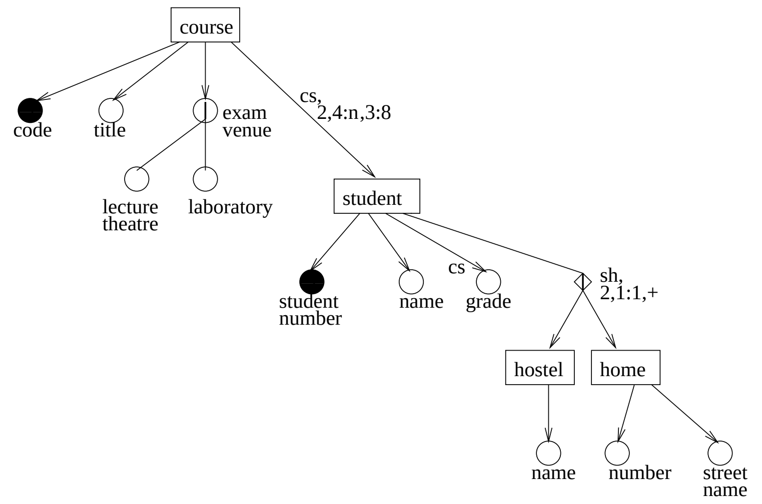


Figure 9: Disjunctive Attribute and Relationship in an ORA-SS Schema Diagram

3.6 References

A reference depicts an object referencing another object, and we say a *reference* object references a *referenced* object. The reference object is not materialized in the nesting relationship within its parent. Rather the referenced object is referenced by the parent of the reference object in some way. Notice that we do not say how the object is referenced. A reference is denoted by a dashed edge between a reference object and a referenced object. The reference and referenced objects can have different labels and different attributes and relationships. References are also used to model recursive and symmetric relationships.

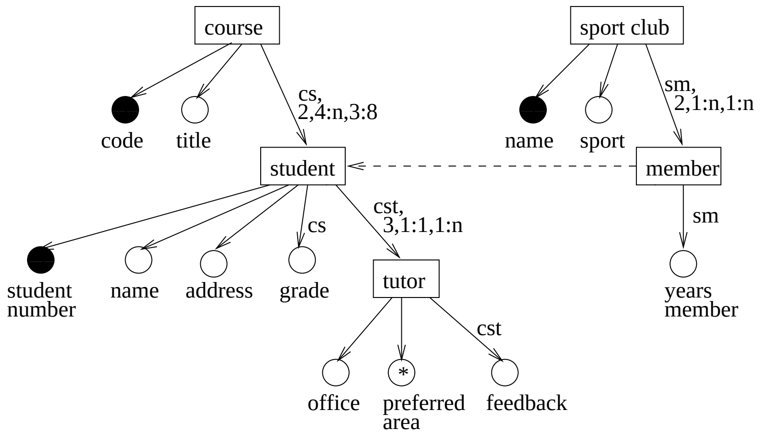


Figure 10: Referencing an object in an ORA-SS Schema Diagram

Example 3.6 Consider the schema in Figure 10. It is an initial model that captures the intended semantics. The object *student* is a child of object *course*, and has attributes *student number*, *name* and *address*. The relationship, *cs*, has attribute *grade*, and there is a ternary relationship between objects *course*, *student* and *tutor*. There is an object *sports club* that has key *name* and attribute *sport*. The relationship, *sm*, between objects *sports club* and *member* has attribute *years member*. There is a reference between *member* (reference object) and *student* (referenced object). If object *student* is accessed through *sports club* then *student number*, *name* and *address* have meaning and can be reached, but the attribute *grade* which is an attribute of the relationship *cs* is not meaningful and cannot be reached. Similarly, it is not possible to reach attribute *years member* when *student* is accessed through *course*.

In fact, we recognize that the materialization of the data suggested by the initial model in Figure 10 is unsatisfactory because it would lead to a lot of redundancy. A better organization is suggested in Figure 11 where *student* and *member* are both reference objects.

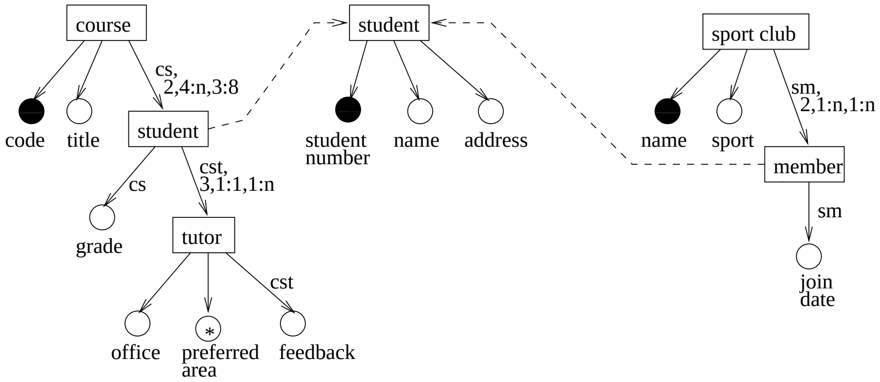


Figure 11: An improved organization over that in Figure 10

We demonstrated in the above example, that not all attributes and relationships are reachable (meaningful) when referencing an object. In fact, only the attributes and relationships that originate at the referenced object are reachable. That is, attributes or relationships that belong to relationships above the referenced object are not reachable (meaningful). In [16], only the roots of trees can be referenced. Because we have defined which attributes and relationships are reachable, any object, not only the root, can be referenced in an ORA-SS schema diagram.

A recursive relationship is modeled using references. There is a reference connecting the recursing object to itself.

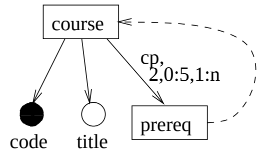


Figure 12: Example of a recursive relationship in ORA-SS Schema Diagrams

Example 3.7 The schema in Figure 12 shows a course with other courses as its prerequisites. Object *prereq* is the reference object, and *course* is the referenced object. Because the relationship *cp* originates at *course*, *prereq*s of *prereq*s are reachable (meaningful).

In a symmetric relationship there is a relationship between two objects, but it is impossible to say that either is nested within the other. A symmetric relationship is also modeled using references.

Example 3.8 Consider the relationship between courses and students, where a course has students and students take courses. This relationship is modeled in Figure 13.

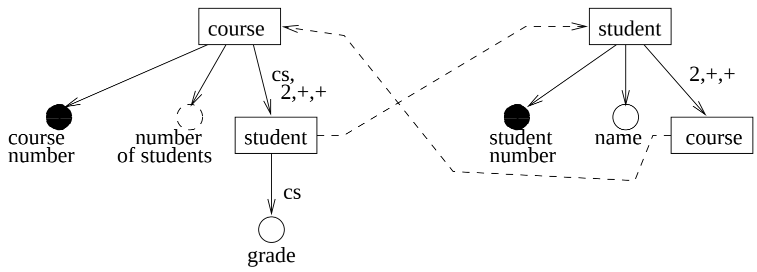


Figure 13: Symmetric relationship in an ORA-SS Schema Diagram

Obviously there are situations where it is advantageous to model a scenario using references and situations where it is disadvantageous. If we assume that nesting in a schema diagram indicates that the corresponding objects in the semi-structured data is nested, and that the data is going to be stored in a repository, then if there is a many-to-many relationship between the parent and child objects then it is better that the parent references the child otherwise there will be a lot of redundant information stored in the repository. If the relationship is a 1-to-many relationship between parent and child, then there is no redundancy when the child is nested, so we would not expect the parent to reference the child in the ORA-SS schema diagram.

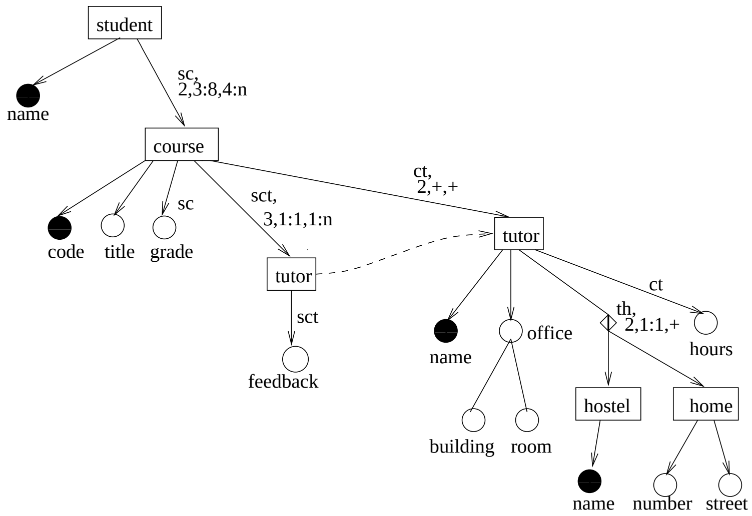


Figure 14: ORA-SS Schema Diagram

Example 3.9 In Figure 14, there is a binary relationship between *student* and *course* and a ternary relationship between *student*, *course* and *tutor*. The *grade* is an attribute of the binary relationship, *sc*, between *student* and *course* and *feedback* is an attribute of the ternary relationship, *sct*. Attribute *hours* is an attribute of the binary relationship, *ct*, between *course* and *tutor*. There is a reference between object *tutor*, which is the child in relationship *sct* and *tutor* which is the child in *ct*.

Figure 15 is an ER diagram representing the information in Figure 14. Notice in this diagram that the ternary relationship is represented as an aggregation, *sc*, and a relationship set, *sct*, between the aggregation, *sc*, and an entity type, *tutor*. Attributes *grade*, *feedback* and *hours* are attributes of relationships *sc*, *sct*, and *ct* respectively. The keys in the ORA-SS schema diagram are the keys in the ER diagram. The reference in the ORA-SS schema diagram is represented as a relationship in the ER diagram.

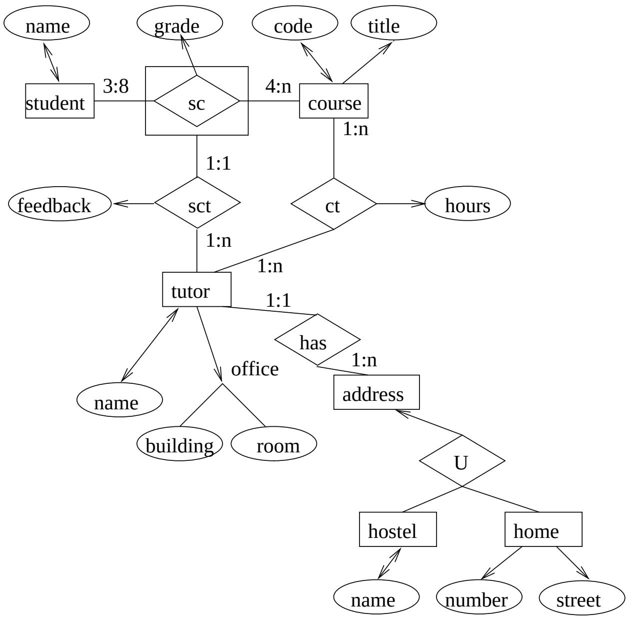


Figure 15: ER diagram of schema in Figure 14

3.7 Functional Dependencies

Functional dependencies model real world constraints, showing how some of the attributes depend on other attributes. Functional dependency diagrams can represent this information when it isn't clear in ER diagrams. Functional dependency diagrams can also be used with ORA-SS schema diagrams to capture the same information. This information is useful when attempting to identify redundancy in the resulting repository. (The functional dependencies of binary relationships can be derived from the schema diagrams, so there is no need to draw functional dependency diagrams for binary relationships.) There are two advantages in having separate functional dependency diagrams: we can express more information, and the information can be expressed without crowding the ORA-SS diagrams.

In functional dependency diagrams the objects are represented by labeled rectangles, the relationships by diamonds labeled only with the relationship name, and the edges carry the cardinality of the objects. The edge labels each have $c_1/\dots/c_n$ where n is the number of functional dependencies between the objects. The c_1 on each edge emanating from a relationship relate to each other and represent one functional dependency, as do the c_2, etc. Each c_1 represents the objects cardinality in that functional dependency. A hyphen ("-") indicates that the object does not take part in the functional dependency. When c_i are all 1, each object is unique (i.e. each object participates in the relationship only once).

Example 3.10 The functional dependency diagram is a separate diagram. Assume that in the scenario being modeled tutors can take many tutorials in one course, each student in a course has one tutor and a tutor can conduct tutorials in only one course. The functional dependencies that model these constraints are:

$$student, course \rightarrow tutor$$
$$tutor \rightarrow course.$$

The functional dependencies between the entities *student*, *course*, and *tutor* are shown in Figure 16. Notice that the symbol "-" is used to show when an object is not participating in a functional dependency.

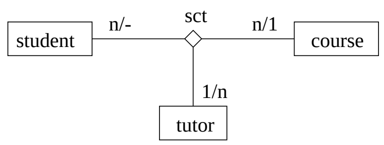


Figure 16: Functional dependency diagram

Given this information, the reason why there are various ways to nest the objects becomes clearer. One tutor is associated with many students and courses, and one course can be associated with many tutors, so there are two nestings:

- nest tutor within course within student, or
- nest course within tutor.

In fact, there will be redundancy if either of these nestings are used as is evidenced by the functional dependencies, so a better organization is to maintain references to the objects rather than materialize them in the nestings.

3.8 Inheritance Hierarchy

Inheritance hierarchies represent common properties of objects. For example, an inheritance hierarchy can be used to show that the properties of a student are a superset of the properties of a person. The inheritance hierarchy is helpful when organizing data storage, because there are various standard methods of efficiently storing objects that are generalizations/specializations of other objects. The methods may not apply directly for semi-structured data.

Example 3.11 Consider the ORA-SS schema diagram in Figure 17(a). Notice that the objects *student*, *faculty* and *tutor* all have the attribute *ssn*, and that *student* and *tutor* also have the attribute *student number*. The inheritance information is drawn on a separate inheritance diagram. The inheritance information for the objects in Figure 17(a) is recorded in the inheritance diagram in Figure 17(b), *tutor* inherits from *student* who inherits from *person*, and *faculty* inherits from *person*.

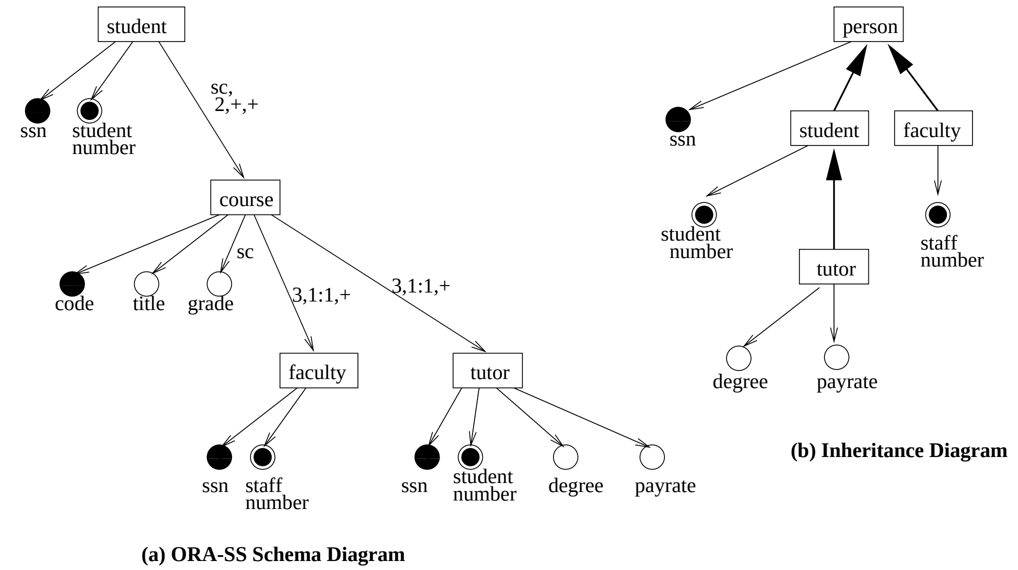


Figure 17: ORA-SS Schema Diagram and Inheritance Diagram

3.9 Constraints

We have demonstrated how to represent functional dependencies, key constraints and participation constraints. Both inclusion dependencies and semantic dependencies can also be represented in the ORA-SS data model. An inclusion dependency shows that the instances of a relationship are a subset of (or are included in) the instances of another relationship. There is an expectation that this constraint holds on tertiary constraints (in fact n-ary where $n > 2$) in the ORA-SS schema diagram. If the constraint does not hold then the objects should not be nested.

Example 3.12 Consider Figure 14, where the relationship between *student*, *course* and *tutor* is an inclusion dependency. That is, the information can be stored in two relations SC(student.name, course.code) and SCT(student.name, course.code, tutor.name) such that SCT[student.name, course.code] $\subseteq$ SC[student.name, course.code], in fact SCT[student.name, course.code] $=$ SC[student.name, course.code].

Semantic dependencies [17] express semantic knowledge or constraints from the real world.

Example 3.13 When we model the date an employee starts with a company, the join date (jdate) can be an attribute of employee (see Figure 18(a)). However, if it is the date that an employee starts in a department, then join date (jdate) must be an attribute of the relationship between employee and department (see Figure 18(b)). In Figure 18(b), we would expect the join date to change whenever the relationship between employee and department changes.

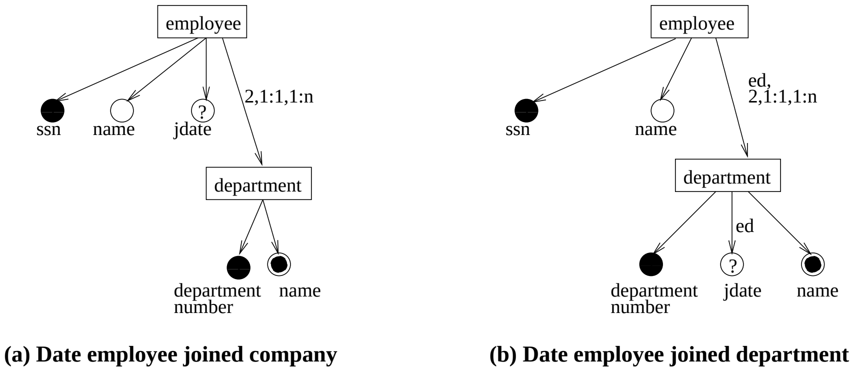


Figure 18: Modeling join date (jdate) in ORA-SS Schema Diagram

3.10 Data Instance

Traditionally, when databases are being designed, the schema is modeled using a conceptual model and the schema of the database is derived from this model. With semi-structured data, the focus is more on the instance and the schema is derived from the instance. The semi-structured data instance can be modeled using an ORA-SS instance diagram. The ORA-SS instance diagram has labeled rectangles for object instances, labeled circles for attribute and their associated data, and the edges represent relationship instances.

Example 3.14 Consider Figure 3(a), where *course* and *student* represent object instances, and the label $Number: CS4221$ on the circle states the name of the attribute, $Number$, and the value of the attribute, $CS4221$.

4 Mapping between ORA-SS diagrams and other representations

Throughout this paper, we have made claims about the ORA-SS data model. In this section we briefly outline how the semantic information in the ORA-SS data model can be derived and used. First, we outline how XML documents and DTDs map to the ORA-SS data model, and then we discuss the mapping of ORA-SS diagrams to a relational database to produce a database with less redundant information than a mapping from a DTD to a relational database.

An XML document maps easily to an ORA-SS instance diagram. The tags with values are the names and values of attributes at leaf nodes and the other tags will be names of objects. The mapping of a DTD to an ORA-SS schema diagram is a little more complex. Elements whose type is `PCDATA` or `ANY` are mapped to attributes. Elements whose type is `EMPTY` are mapped to composite attributes, with the attributes in the related attribute list as the components. Attributes with attribute type `ID` are mapped to primary keys and attributes with attribute type `IDREF` indicate a reference object with a reference to a referenced object. The cardinality of the attributes can also be mapped from the DTD. All other elements in the DTD can be mapped to objects and the nesting relationships are derived from the nesting in the DTD. The advantage of such a mapping is that no information is lost in the mapping. There is ordering imposed on the elements in a DTD. We can capture this information in the ORA-SS schema diagram, but we would usually choose not to and then add ordering manually where it is important. DTDs have no concept of relationship, so there is no way of expressing the degree of a relationship, the cardinality of relationships, or whether an attribute belongs to a relationship or object. ORA-SS schema diagrams need to be annotated manually with this information. It is also not possible to derive functional dependency diagrams or inheritance diagrams from DTDs.

There are several algorithms for mapping DTDs to relational databases [10, 12, 27]. The algorithms are based around a common theme, where they map elements to relations, and some relationships to separate relations depending on their cardinality. The authors of [15] recognized that by including some semantic information that is hidden in the DTD, databases that enforce some important constraints can be produced. In a DTD it is impossible to distinguish object attributes from relationship attributes so they are all simply modeled as the attributes of the object. Similarly because it is not possible to establish the degree of the relationships in a DTD, they are all modeled as binary relationships. A mapping from the ORA-SS data model can be based on the current mappings from DTDs to relational databases, but the work needs to be extended. One simple extension would involve using the degree of the relationships and the information about whether attributes belong to objects or relationships. An n-ary relationship can be modeled as a relation, maintaining this semantic information in the database. Relationships with attributes can then be modeled as relations.

In fact, there is a lot of fragmentation when semi-structured data is mapped to relational databases. Our investigations to date confirm that semi-structured data maps more naturally to object-relational databases. A further extension would involve defining something like a normalization algorithm for the ORA-SS data model to further reduce the redundancy of data in the resulting object-relational database.

5 Related Work

The ORA-SS data model extends semi-structured data models that have been proposed in the literature. These models commonly represent semi-structured data as directed labeled graphs [1, 4, 28]. Of the models proposed, OEM (Object Exchange Model) [2, 13] is a representative example. In OEM, entities are represented by objects, each object has an object identifier and each object is either atomic or complex, i.e. the value of the object is atomic or a set of references respectively. OEM is a simple and flexible model, for representing semi-structured data, which facilitates data exchange, conversion and integration of heterogeneous data [25, 26, 8]. However, in OEM it is not possible to express the semantic information that is needed to design an efficient non-redundant data repository, or for query formulation and optimization.

In [30] the author states that applying well-developed theoretical and practical techniques for designing semi-structured data is an important issue in semi-structured data research. To apply techniques like normalization, effectively, it is necessary to know the cardinality of objects in relationships, the degree of n-ary relationships and whether attributes are attributes of objects or attributes of relationships. In OEM, the relationships between objects are not made specific. Like in the object-oriented data model, the inter-object references may introduce maintenance problems as a result of redundancy [18]. In contrast, in the ORA-SS data model, the relationships between objects and the properties of these relationships are stated explicitly. The work in [16], that provides a way to normalize semi-structured repositories should be easy to extend to the ORA-SS data model. The ORA-SS data model is a foundation to which well-developed theoretical and practical techniques for designing semi-structured data can be applied.

In [23], the authors highlight that the OEM model does not provide a notion of a canonical path to a node, which makes query formulation and optimization difficult, if not impossible. Various models have been proposed that address this problem. They either define a schema for the semi-structured data [5, 8, 14, 23], or have some constraints on the data graph [6, 7]. Some research [14, 22, 24] describes ways to extract a schema or infer a schema from a semi-structured data instance. Typically their schema formalization is based on logic or graph theory, and focuses either on describing patterns in the data graph or specifying the possible existence of edges with a given property emanating from a node. The ORA-SS schema diagram describes a canonical path to a node, and is based on the well-understood entity relationship model, allowing us to capture key constraints, participation constraints, domain constraints etc. The nested structure of the underlying semi-structured data is reflected in the ORA-SS data model, and in contrast to the ER model the attributes may not have the same structure in all instances of an object. The functional dependency diagrams in the ORA-SS data model can be used to find multi-valued dependencies, using the operations defined in [19].

6 Conclusion and Future Work

In the current data models for semi-structured data it is not possible to model the kind of information that is traditionally needed when organizing data storage, designing repositories, or defining views. In this paper, we have presented the ORA-SS data model in which this information can be represented. The data model consists of four diagrams: the schema diagram, the instance diagram, the functional dependency diagram and the inheritance diagram. The schema diagram is the richest diagram in the ORA-SS data model. The most important features in this diagram are that the nested structure of the data is inherent in the schema, a distinction is made between objects, relationships and attributes, and the relationship attributes are distinguished from the object attributes. We have defined which attributes are reachable by an object that references another object. The instance diagram shows the instances of objects, relationships and attributes. The functional dependency diagram shows the cardinality of objects in relationships. The inheritance diagram shows commonality in attributes between objects. We have shown how various concepts can be represented using an ORA-SS data model and presented some examples that are more complicated to model due to the implicit nested structure enforced in the semi-structured data model. Although the examples in this paper are quite simple they are representative of more complicated situations that people represent using semi-structured data.

The work presented in this paper can be used as the basis for other work in the semi-structured data field. We list some of the possibilities here: the ORA-SS data model can be used to recognize if there is redundancy in a semi-structured data repository; this information can be used to define normal forms for semi-structured repositories; the semantic information represented can be used to design efficient storage organization and access paths to the data; the data model can be used to identify which views are plausible and how the views can be updated; approximate query answering uses the schema to find related answers; the data model provides a user-friendly way to visualize the instance and schema of a semi-structured data store; the ORA-SS provides a standard way of representing schemas which can be used for data integration. In order to have a simple data model, we omitted to include some concepts that are necessary in such a model. These concepts can easily be added to this model, e.g., types and enumerated types can be represented as labels of leaves. Using the data model to model real systems will show what extensions are necessary from a practical perspective.

There are many directions this research can go from here. Data integration is an important area. A large part of data integration involves finding equivalences or matches between two or more schema. The problem of finding equivalences between diagrams is very complex. It is easier to find equivalences automatically in a standard textual description. We have already started describing a syntax for the textual description. Building tools to create web pages and to deal with semi-structured data is a very active research area. The ORA-SS data model is the ideal notation on which to base such a tool, because the notation is simple and yet it describes the semantics that you need when defining the structure of the web page and how the data should be stored.

References

[1] Serge Abiteboul. Querying semi-structured data. In *Proceedings of the 5th International Conference on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 1997.

[2] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[3] Tim Bray, Jean Paoli, C.M Sperberg-McQueen, and Eve Maler (eds). Extensible markup language (xml) 1.0, second edition. http://www.w3.org/TR/2000/REC-xml-20001006, 6 Oct, 2000.

[4] Peter Buneman. Semistructured data. In *Proceedings of the 6th ACM Symposium on Principles of Database Systems*, pages 117–121. ACM Press, 1997.

[5] Peter Buneman, Susan B. Davidson, Mary F. Fernandez, and Dan Suciu. Adding structure to unstructured data. In *Proceedings of the 6th International Database Theory Conference Database Theory (ICDT'97) Database Theory - ICDT '97*, volume 1186 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 1997. There is also a VLDB journal 2000 version.

[6] Peter Buneman, Wenfei Fan, and Scott Weinstein. Path constraints in semistructured and structured databases. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 129–138. ACM Press, 1998. The same authors have a PODS99 paper entitled "Interaction between Path and Type Constraints".

[7] A. Cali, D. Calvanese, and M. Lenzerini. Semistructured data schemas with expressive constraints. In *Proceedings of the 7th International Workshop on Knowledge Representation meets Databases (KRDB'99), Berlin, Germany, August 21*, pages 3–16, 2000. The same authors have a SEBD paper entitled "Local constraint in semistructured data schemas".

[8] Sophie Cluet, Claude Delobel, Jerome Simeon, and Katarzyna Smaga. Your mediators need data conversion! In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 177–188. ACM Press, 1998.

[9] Lotus Development Corporation. Lotus development corporation. http://www.lotus.com/.

[10] A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with STORED. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 431–442. ACM Press, 1999.

[11] Mary Fernandez, Daniela Florescu, Jaewoo Kang, Alon Levy, and Dan Suciu. Catching the boat with Strudel: experiences with a web-site management system. In *SIGMOD*, pages 414–425, 1998.

[12] D. Florescu and D. Kossmann. Storing and querying xmldata using an rdbms. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[13] R. Goldman, J. McHugh, and J. Widom. From semistructured data to XML: Migrating the lore data model and query language. In *Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99)*, pages 25–30, 1999.

[14] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 436–445. Morgan Kaufmann, 1997.

[15] Dongwon Lee and Wesley W. Chu. Constratints-preserving transformation from xml document type definition to relational schema. In *Proceedings 19th International Conference on Conceptual Modeling (ER'00)*, volume 1920 of *Lecture Notes in Computer Science*, pages 323–338. Springer, 2000.

[16] Sin Yeung Lee, Mong Li Lee, Tok Wang Ling, and Leonid A. Kalinichenko. Designing good semi-structured databases. In *Proceedings 18th International Conference on Conceptual Modeling (ER'99)*, volume 1728 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 1999.

[17] Tok Wang Ling and Mong Li Lee. Realtional to entity relationship schema translation using semantic and inclusion dependencies. *Journal of Integrated Computer-Aided Engineering*, 2(2):125–145, 1995.

[18] Tok Wang Ling and Pit Koon Teo. A normal form object-oriented entity relationship diagram. In *Proceedings 13th International Conference on Conceptual Modeling (ER'94)*, volume 881 of *Lecture Notes in Computer Science*, pages 241–258. Springer, 1994.

[19] Mengchi Liu and Tok Wang Ling. A data model for semistructured data with partial and inconsistent information. In *Proceedings 6th International Conference on Extending Database Technology (EDBT 2000)*, volume 1777 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2000.

[20] B. Luddscher, R. Himmervder, G. Lausen, W. May, and C. Schlepphorst. Managing semistructured data with FLORID: A deductive object-oriented perspective. *Information Systems*, 23(8):589–613, 1998.

[21] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, 1997.

[22] T. Milo and D. Suciu. Type inference for queries on semistructured data. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 215–226. ACM Press, 1999.

[23] Tova Milo and Sagit Zohar. Using schema matching to simplify heterogeneous data translation. In *Proceedings of 24rd International Conference on Very Large Data Bases (VLDB'98)*, pages 122–133. Morgan Kaufmann, 1998.

[24] Svetlazar Nestorov, Serge Abiteboul, and Rajeev Motwani. Extracting schema from semistructured data. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 295–306. ACM Press, 1998.

[25] Yannis Papakonstantino, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, pages 251–260. IEEE Computer Society, 1995.

[26] Yannis Papakonstantinou, Serge Abiteboul, and Hector Garcia-Molina. Object fusion in mediator systems. In *Proceedings of 22nd International Conference on Very Large Data Bases (VLDB'96)*, pages 413–424. Morgan Kaufmann, 1996.

[27] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 79–90. Morgan Kaufmann, 1999.

[28] Dan Suciu. Semistructured data and XML. In *Proceedings of 5th International Conference on Foundations of Data Organization (FODO '98)*, 1998.

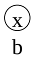
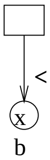
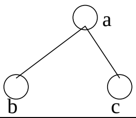
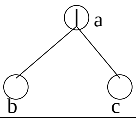
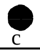
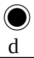
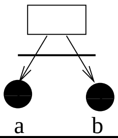
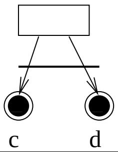
[29] Tamino - An Internet Database System. http://www.tamino.com/.

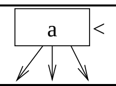
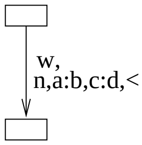
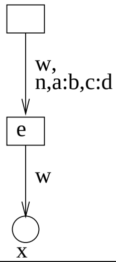
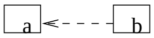
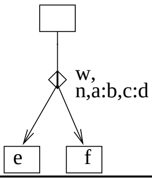
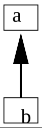
[30] J. Widom. Data management for xml: Research directions. *IEEE Data Engineering Bulletin*, 22(3):44–52, 1999.

[31] Xml schema. http://www.w3.org/XML/Schema.

Appendix

The following tables summarize the notation of ORA-SS diagrams.

notation	description
	object with name a
	attribute b, where x represents the cardinality, ? is 0 or 1, + is 1 or more, * is 0 or more, and the default value for x is ? (0 or 1).
	attribute b where the ordering of the value of the attributes is important. x is either + or *, and the default value is *.
	composite attribute a with component attributes b and c
	disjunctive attribute a is either b or c
	identifier/primary key c
	candidate key d
	composite identifier/primary key
	composite candidate key
	derived attribute
	attribute with unknown structure or whose structure is heterogeneous

notation	description
	the ordering on the attributes of object a is important
	relationship with name w, of degree n, where the participation of the parent has minimum a and maximum b, and the child has minimum c and maximum d, and the ordering of the objects is important. The default degree is 2, default parent cardinality is $1 : m$, default child cardinality is $0 : n$, and default on ordering is no ordering.
	attribute x belongs to relationship w. The default (without label w on the edge) shows that attribute x belongs to object e.
	reference object a references referenced object b
	disjunctive relationship: either object e or object f
	b inherits from a (inheritance diagram)
n/m	cardinality of objects in relationships (functional dependency diagram)