Validating Semistructured Data Using OWL

Yuan Fang Li[1,⋆], Jing Sun[2], Gillian Dobbie[2], Jun Sun[1], and Hai H. Wang[3,⋆⋆]

[1] School of Computing, National University of Singapore, Singapore
{liyf, sunj}@comp.nus.edu.sg
[2] Department of Computer Science, The University of Auckland, New Zealand
{j.sun, gill}@cs.auckland.ac.nz
[3] Department of Computer Science, University of Manchester
hai.wang@cs.manchester.ac.uk

Abstract. Semistructured data has become prevalent in both web applications and database systems. This rapid growth in use makes the design of good semistructured data essential. Formal semantics and automated reasoning tools enable us to reveal the inconsistencies in a semistructured data model and its instances. The Object Relationship Attribute model for Semistructured data (ORA-SS) is a graphical notation for designing and representing semistructured data. This paper presents a methodology of encoding the semantics of ORA-SS in the Web Ontology Language (OWL) and automatically validating the semistructured data design using the OWL reasoning tool - RACER. Our methodology provides automated consistency checking of an ORA-SS data model at both the schema and instance levels.

Keywords: Semistructured Data, Semantic Web, OWL, Formal Verification.

1 Introduction

Semistructured data has become prevalent in both web applications and database systems. It acts as a hinge technology between the data exchanged on the web and the data represented in a database system. This rapid growth in use makes the design of good semistructured data essential. Many data modeling languages [1, 3, 5, 10] for semistructured data have been introduced to capture more detailed semantic information. The Object Relationship Attribute model for Semistructured data (ORA-SS) [4, 9] is a semantic enriched graphical notation for designing and representing semistructured data [8, 9, 11]. The ORA-SS data model not only reflects the nested structure of semistructured data, but also distinguishes between object classes, relationship types and attributes. The main advantages of ORA-SS over other data models is its ability to express the degree of an n-ary relationship type, and distinguish between the attributes of relationship types and the attributes of object classes. This semantic information is

⋆ The author would like to thank Singapore Millennium Foundation (SMF) for the financial support.

⋆⋆ This work was supported in part by the CO-ODE project funded by the UK Joint Information Services Committee, the HyOntUse Project (GR/S44686) funded by the UK Engineering and Physical Science Research Council.

J.X. Yu, M. Kitsuregawa, and H.V. Leong (Eds.): WAIM 2006, LNCS 4016, pp. 520–531, 2006.

essential, even crucial for semistructured data representation and management, but it is lacking in other existing semistructured data modeling notations.

A major concern in designing a good semistructured data model using ORA-SS for a particular application is to reveal any possible inconsistencies at both the schema and instance levels. Inconsistencies at the schema level arise if a customized ORA-SS schema model does not conform to the ORA-SS notation. Inconsistencies at the instance level arise if an instance document is not consistent with its ORA-SS schema definition. For example, an inconsistency that might arise at the schema level is the specification of a ternary relationship between only two object classes. An inconsistency that might arise at the instance level is a many to many relationship between elements when a one to many relationship is specified in the schema. These two aspects of validation are essential in the semistructured data design process. Thus, the provision of formal semantics and automated reasoning support for validating ORA-SS semistructured data modeling is very beneficial.

Recent research on the World Wide Web has extended to the semantics of web content. More meaningful information is embedded into the web content, which makes it possible for intelligent agent programs to retrieve relevant semantic as well as structural information based on their requirements. The Semantic Web [2] approach proposed by the World Wide Web Consortium (W3C) attracts the most attention. It is regarded as the next generation of the web. The Ontology Web Language (OWL) is an ontology language for the Semantic Web. OWL can provide not only the structural information of the web content but also meaningful semantics for the information presented. The aim of this paper is to encode the semantics of the ORA-SS notation into the Web Ontology Language (OWL) and automatically verify the semistructured data design using the OWL reasoning tool RACER [6].

The reason that we chose OWL to fulfil our goal is due to the nature of the semistructured data and its strong connections to web technologies. Semistructured data is typically represented using eXtensible Markup Language (XML). XML is a commonly used exchange format in many web and database applications. The introduction of the Semantic Web is to overcome the structure-only information of XML, and to provide deeper semantic meanings to the data. The ORA-SS data model is a semantically enriched data modeling language for describing semistructured data. From the point of capturing more semantic information in semistructured data, OWL and ORA-SS are two approaches that fulfil the same goal, where the former is rooted from the web community and the latter has its basis in the database community. We believe that Semantic Web and its reasoning tools can contribute to the verification phase of the semistructured data design.

In this paper, we propose a methodology to validate semistructured data design using OWL and its reasoner RACER. Firstly, we define an ontology model of the ORA-SS data modeling language in OWL. It provides a rigorous semantic basis for the ORA-SS graphical notation and enable us to represent any ORA-SS data model and its instances in OWL. Furthermore, RACER is used to perform the automated verification of the correctness in a semistructured data design. Our approach is able to provide automatic consistency checking on large semistructured data models and their instances.

The remainder of the paper is organized as follows. Section 2 briefly introduces the background knowledge for the semistructured data modeling language ORA-SS,

Semantic Web ontology language OWL and its reasoning tool RACER. Section 3 presents OWL semantics of the ORA-SS notation and its data models. Section 4 demonstrates a case study on a complete ontology reasoning process for verifying semistructured data design. Examples of both class-level reasoning and instance-level reasoning are presented. Finally, Section 5 concludes the paper.

2 Background

2.1 The ORA-SS Data Modeling Language

The Object Relationship Attribute model for Semistructured data (ORA-SS) data modeling language [4, 9] consists of four basic concepts: object class, relationship type, attribute and reference. It represents these concepts through four diagrams: schema diagram, instance diagram, functional dependency diagram and inheritance diagram. We will focus on the schema and instance diagram in this paper since they are sufficient for our purposes. A full description of the ORA-SS data modeling language can be found in [4, 9].

- An object class is like an entity type in an ER diagram, a class in an object-oriented diagram or an element in an XML document. The object classes are represented as labeled rectangles in an ORA-SS diagram.
- A relationship type represents a nesting relationship among object classes. It is described as a labeled edge by a tuple (`name`, `n`, `p`, `c`), where the `name` denotes the name of relationship type, integer `n` indicates degree of relationship type, `p` represents participation constraint of parent object class in relationship type and `c` represents participation constraint of child object class in relationship type.
- Attributes represent properties and are denoted by labeled circle. An attribute can be a key attribute which has a unique value and represented as a filled circle. Other types of attributes include single valued attribute, multi-valued attribute, required attribute,

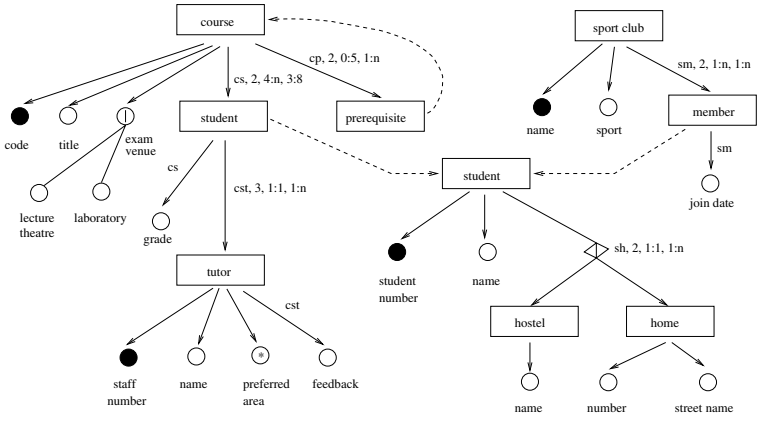

Fig. 1. The ORA-SS Schema Diagram of a Course-Student data model

composite attribute, etc. An attribute can be a property of an object class or a property of a relationship type.
– An object class can reference another object class to model recursive and symmetric relationships, or to reduce redundancy especially for many-to-many relationships. It is represented by a labeled dashed edge.

For the design of semistructured data, an ORA-SS schema diagram constrains the relationships, participations and cardinalities among the instances of the object classes in a semistructured data model. For example, Fig. 1 represents an ORA-SS schema diagram of a `Course-Student` data model. In the diagram, each `course` has `code`, `title`, `exam venue` as its attributes. A relationship type `cs`, which indicates the relationship between a `course` object class and a `student` object class is binary, and each course consists of 4 to many students and each student can select 3 to 8 courses. The `student` object class in the `cs` relationship type has a reference pointing to its complete definition. The `grade` attribute is an attribute belonging to the `cs` relationship type. Based on the above schema definition, two levels of validation can be carried out. Firstly, consistency checking can be performed to determine whether the defined schema model is correct with respect to the ORA-SS language. Secondly, consistency checking can be performed to determine whether a particular instance of semistructured data satisfies the defined ORA-SS schema model. Hence automated tool support for validating the consistency in an ORA-SS data model would be highly desirable.

2.2 Semantic Web – OWL and RACER

Description logics are logical formalisms for representing information about knowledge in a particular domain. It is a subset of first-order predicate logic and is well-known for the trade-off between expressivity and decidability.

The Web Ontology Language (OWL) [7] is the de-facto ontology language for the Semantic Web. It consists of three increasingly expressive sublanguages: OWL Lite, DL and Full. OWL DL is very expressive yet decidable. As a result, core inference problems, namely concept subsumption, consistency and instantiation, can be performed fully automatically. In OWL, conceptual entities are organized as classes in hierarchies. Individual entities are grouped under classes and are called instances of the classes. Classes and individuals can be related by properties. We will be using a synatx similar to that presented in [7].

RACER, the **R**enamed **A**Box and **C**oncept **E**xpression **R**easoner [6], is a reasoning engine for ontologies languages DAML+OIL and OWL. It implements a TBox and ABox reasoner for the description logic $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D})^-$ [6]. It is fully automated for reasoning over OWL Lite and DL ontologies.

3 Modeling ORA-SS Data Design Models in OWL

In this section, we show the modeling of ORA-SS schema and instance diagrams as OWL ontologies in three parts. Firstly, we define the ORA-SS ontology in Section 3.1, which contains the OWL definitions of essential ORA-SS concepts. Secondly, in the next 3 subsections, we show how individual schema diagram ontology can be constructed based on the ORA-SS ontology. Finally, in Section 3.5, we show how instance diagrams can be represented in OWL.

Our modeling approach can be regarded as a methodology for creating the OWL representation of ORA-SS diagrams. By strictly following this methodology, a lot of potential modeling errors can be avoided, which will become more evident as we present the approach below. To effectively illustrate the modeling approach, the schema diagram in Fig. 1 is used as a running example.

3.1 The ORA-SS Ontology

The ORA-SS ontology[1] contains the OWL definitions for ORA-SS concepts such as object class, relationship type, attribute, etc. We will model these definitions as OWL classes. The basic assumption here is that all named OWL classes are by default mutually disjoint, which is implied in the ORA-SS diagrams. Essential properties are also defined in the ontology. This ontology, with a namespace of `ora-ss`, can be used later to define ontologies for ORA-SS schema diagrams.

Entities. As each object class and relationship type can be associated with attributes and other object classes or relationship types, we define an OWL class *ENTITY* to represent the super class of both object class and relationship type. The OWL class structure is shown as follows.

$ENTITY \sqsubseteq \top$ $\quad$ $ATTRIBUTE \sqsubseteq \top$

$OBJECT \sqsubseteq ENTITY$ $\quad$ $ENTITY \sqcap ATTRIBUTE = \bot$

$RELATIONSHIP \sqsubseteq ENTITY$ $\quad$ $OBJECT \sqcap RELATIONSHIP = \bot$

It may not seem very intuitive to define relationship types as OWL classes. In ORA-SS, relationship types are used to relate various object classes and relationship types, it might be natural to model relationship types as OWL properties. However, there are two reasons that we decide to model relationship types as OWL classes. Firstly, the domain of ORA-SS relationship types can be relationship types themselves, which describes the relationships of ternary and more. Secondly, classes and properties in OWL DL are disjoint. In our model, a relationship type class consists of instances which are actually pointers to the pairs of object classes or relationship types that this relationship relates.

As ORA-SS is a modeling notation for semistructured data, we need to cater to unstructured data. We define a subclass of *ATTRIBUTE* called *ANY* as a place holder to denote any unstructured data appearing in a model. In ORA-SS, a composite attribute is an attribute composed of other attributes. We also define it as a subclass of *ATTRIBUTE*.

$ANY \sqsubseteq ATTRIBUTE$ $\quad$ $CompositeAttribute \sqsubseteq ATTRIBUTE$

$ANY \sqcap CompositeAttribute = \bot$

Properties. A number of essential properties are defined in the `ora-ss` ontology.

Properties Among Entities

In ORA-SS, object classes and relationship types are inter-related to form new relationship types. As mentioned above, since we model relationship types as OWL classes, we need additional properties to connect various object classes and relationship types.

[1] Available at `http://www.comp.nus.edu.sg/~liyf/ora-ss/ora-ss.owl`

Firstly, this is accomplished by introducing two object-properties, *parent* and *child*, which map a *RELATIONSHIP* to its domain and range *ENTITY*s. The following statements define the domain and range of *parent* and *child*. As in ORA-SS, the domain of a relationship (*parent*) can be either an object class or another relationship type, i.e., an *ENTITY*. The range (*child*) must be an *OBJECT*. These two properties are functional as one relationship type has exactly one domain and one range node. Moreover, we assert that only relationship types can have parents and child but object classes cannot.

$\geq 1\ parent \sqsubseteq RELATIONSHIP$ $\geq 1\ child \sqsubseteq RELATIONSHIP$

$\top \sqsubseteq \forall\ parent.ENTITY$ $\top \sqsubseteq \forall\ child.OBJECT$

$\top \sqsubseteq \leq 1\ parent$ $\top \sqsubseteq \leq 1\ child$

$RELATIONSHIP \sqsubseteq \forall\ parent.ENTITY$ $RELATIONSHIP \sqsubseteq \forall\ child.OBJECT$

Secondly, we define two more object-properties: *p-ENTITY-OBJECT* and *p-OBJECT-ENTITY*. These two properties are inverse of each other and they serve as the super properties of the properties that are to be defined in later ontologies of ORA-SS schema diagrams. Those properties will model the restrictions imposed on the relationship types.

The domain and range of *p-ENTITY-OBJECT* are *ENTITY* and *OBJECT*, respectively. Since the two properties are inverse, the domain and range of *p-OBJECT-ENTITY* can be deduced.

$p\text{-}OBJECT\text{-}ENTITY = (^{-}p\text{-}ENTITY\text{-}OBJECT)$

$\geq 1\ p\text{-}ENTITY\text{-}OBJECT \sqsubseteq ENTITY$ $\geq 1\ p\text{-}OBJECT\text{-}ENTITY \sqsubseteq OBJECT$

$\top \sqsubseteq \forall\ p\text{-}ENTITY\text{-}OBJECT.OBJECT$ $\top \sqsubseteq \forall\ p\text{-}OBJECT\text{-}ENTITY.ENTITY$

$ENTITY \sqsubseteq \forall\ p\text{-}ENTITY\text{-}OBJECT.OBJECT$ $OBJECT \sqsubseteq \forall\ p\text{-}OBJECT\text{-}ENTITY.ENTITY$

Properties Between Entities and Attributes

First of all, we define an object-property *has-ATTRIBUTE*, whose domain is *ENTITY* and range is *ATTRIBUTE*. Every *ENTITY* must have *ATTRIBUTE* as the range of *has-ATTRIBUTE*.

$\geq 1\ has\text{-}ATTRIBUTE \sqsubseteq ENTITY$ $ENTITY \sqsubseteq \forall\ has\text{-}ATTRIBUTE.ATTRIBUTE$

$\top \sqsubseteq \forall\ .has\text{-}ATTRIBUTE.ATTRIBUTE$

For modeling the ORA-SS candidate and primary keys, we define two new object properties that are sub-properties of *has-ATTRIBUTE*. We also make the property *has-primary-key* inverse functional and state that each *ENTITY* must have at most one primary key. Moreover, we restrict the range of *has-candidate-key* to be *ATTRIBUTE*.

$has\text{-}candidate\text{-}key \sqsubseteq has\text{-}ATTRIBUTE$ $has\text{-}primary\text{-}key \sqsubseteq has\text{-}candidate\text{-}key$

$\top \sqsubseteq \forall\ has\text{-}candidate\text{-}key.ATTRIBUTE$ $\top \sqsubseteq \leq 1\ has\text{-}primary\text{-}key^{-}$

$ENTITY \sqsubseteq \leq 1\ has\text{-}primary\text{-}key$

3.2 Object Classes

In this subsection, we present how ORA-SS object classes in a schema diagram are represented in OWL. Moreover, we will discuss how object class referencing is modeled.

Example 1. The schema diagram in Fig. 1 contains a number of object classes [2].

$course \sqsubseteq OBJECT$ $\quad$ $tutor \sqsubseteq OBJECT$

$student \sqsubseteq OBJECT$ $\quad$ $sport\_club \sqsubseteq OBJECT$

$hostel \sqsubseteq OBJECT$ $\quad$ $home \sqsubseteq OBJECT$

$\ldots$ $\quad$ $\ldots$

Referencing. In ORA-SS, an object class can reference another object class to refer to its definition, which we say that a *reference* object class references a *referenced* object class. In our model, we model the *reference* object class a sub class of the *referenced* object class. If the two object classes are of the same name, the reference object class is renamed. By doing so, we ensure that all the attributes and relationship types of the referenced object classes are reachable (meaningful). Note that there are no disjointness axioms among the reference and referenced object classes.

Example 2. In Fig. 1, the object class student is referenced by object classes student and member. Hence, we rename the reference student to *student*_1 and add the following axioms in to the model.

$student \sqsubseteq OBJECT$ $\quad$ $student\_1 \sqsubseteq student$ $\quad$ $member \sqsubseteq student$

3.3 Relationship Types

In this subsection, we present the details of how ORA-SS relationship types are modeled in OWL. Various kinds of relationship types, such as disjunctive relationship types and recursive relationship types are also modeled. We begin with an example to show the basic modeling of relationship types.

For example, Fig. 1 contains 5 relationship types, namely *cs*, *sh*, *sm*, *cp* and *cst*. The relationship type *cs* is bound by the *parent*/*child* properties as follows. We use both allValuesFrom and someValuesFrom restriction to make sure that only the intended class can be the parent/child class of *cs*.

$cs \sqsubseteq \forall\, parent.course$ $\quad$ $cs \sqsubseteq \forall\, child.student\_1$

$cs \sqsubseteq \exists\, parent.course$ $\quad$ $cs \sqsubseteq \exists\, child.student\_1$

Auxiliary Properties. As discussed in Section 3.1, for each ORA-SS relationship type we define two object-properties that are the inverse of each other.

Example 3. Take *cs* as an example, we construct two object-properties: *p-course-student* and *p-student-course*. Their domain and range are also defined.

$p\text{-}student\text{-}course = (^{-}p\text{-}course\text{-}student)$ $\quad$ $p\text{-}student\text{-}course \sqsubseteq p\text{-}OBJECT\text{-}ENTITY$

$p\text{-}course\text{-}student \sqsubseteq p\text{-}ENTITY\text{-}OBJECT$

$\geq 1\ p\text{-}course\text{-}student \sqsubseteq course$ $\quad$ $\geq 1\ p\text{-}student\text{-}course \sqsubseteq student\_1$

$\top \sqsubseteq \forall\, p\text{-}course\text{-}student.student\_1$ $\quad$ $\top \sqsubseteq \forall\, p\text{-}student\text{-}course.course$

[2] For brevity reasons, the class disjointness statements are not shown from here and onwards.

Participation Constraints. One of the important advantages that ORA-SS has over XML Schema language is the ability to express participation constraints for parent/child nodes of a relationship type. This ability expresses the cardinality restrictions that must be satisfied by ORA-SS instances.

Using the terminology defined previously, ORA-SS parent participation constraints are expressed using cardinality restrictions in OWL on a sub-property of *p-ENTITY-OBJECT* to restrict the parent class *Prt*. Child participation constraints can be similarly modeled, using a sub property of *p-OBJECT-ENTITY*.

Example 4. In Fig. 1, the constraints captured by the relationship type `cs` state that a `course` must have at least 4 students; and a `student` must take at least 3 and at most 8 courses. The following axioms are added to the ontology. The two object-properties defined above capture the relationship type between `course` and `student`.

$$course \sqsubseteq \forall\, p\text{-}course\text{-}student.student\_1 \qquad student\_1 \sqsubseteq \forall\, p\text{-}student\text{-}course.course$$
$$course \sqsubseteq \geq 4\ p\text{-}course\text{-}student \qquad student\_1 \sqsubseteq \geq 3\ p\text{-}student\text{-}course$$
$$student\_1 \sqsubseteq \leq 8\ p\text{-}student\text{-}course$$

Disjunctive Relationship Types. In ORA-SS, a disjunctive relationship type is used to represent disjunctive object classes, where only one object can be selected from a set of object classes. To model this in OWL, we will create a dummy class as the *union* of the disjoint classes and use it as the range of the object-property representing the relationship type. Together with the cardinality constraint that exactly one individual of the range can be selected, the disjunctive relationship type can be precisely modeled.

Example 5. In Fig. 1, *sh* is a disjunctive relationship type where a student must live in exactly one hostel or one home, but not both. We use the following OWL statements to model this situation. Note that *p-student-sh* is an object-property that maps *student* to its range class *home_hostel*, which is the union of *hostel* and *home*.

$$hostel \sqsubseteq OBJECT \qquad home \sqsubseteq OBJECT \qquad hostel \sqcap home = \bot$$
$$home\_hostel = hostel \sqcup home \qquad \top \sqsubseteq \forall\, p\text{-}student\text{-}sh.home\_hostel \qquad \geq 1\ p\text{-}student\text{-}sh \sqsubseteq student$$

Given the above definitions, the disjunctive relationship type *sh* in the schema diagram can be modeled as follows.

$$student \sqsubseteq \forall\, p\text{-}student\text{-}sh.home\_hostel \qquad student \sqsubseteq = 1\ p\text{-}student\text{-}sh$$

3.4 Attributes

The semantically rich ORA-SS model notation defines many kinds of attributes for object classes and relationship types. These include candidate and primary keys, single-valued and multi-valued attributes, required and optional attributes, etc. In this subsection, we will discuss how these attributes can be modeled.

Example 6. The schema diagram in Fig. 1 includes attributes such as *code*, *title* and *exam_venue*, which are all sub classes of *ATTRIBUTE*.

Modeling of Various Definitions. As OWL adopts the Open World Assumption [7] and an ORA-SS model is closed, we need to find ways to make the OWL model capture the intended meaning of the original diagram. The following are some modeling *tricks*.

– For each *ENTITY*, we use an allValuesFrom restriction on *has-ATTRIBUTE* over the union of all the *ATTRIBUTE* classes this *ENTITY* has in the ORA-SS model to denote the complete set of attributes it holds.

  Example 7. In the running example, the object class *student* has student number and name as its attributes.

  $$student \sqsubseteq \forall\, has\text{-}ATTRIBUTE.(student\_number \sqcup name)$$

– Each entity (object class or relationship type) can have a number of attributes. For each of the entity-attribute pairs in an ORA-SS schema diagram, we define an object-property, whose domain is the entity and range is the attribute.

  Example 8. In Fig. 1, the object class *sport_club* has an attribute *name*. It can be modeled as follows.

  $$\geq 1\ has\text{-}sport\_club\text{-}name \sqsubseteq sport\_club \qquad has\text{-}sport\_club\text{-}name \sqsubseteq has\text{-}ATTRIBUTE$$
  $$\top \sqsubseteq \forall\, has\text{-}sport\_club\text{-}name.name \qquad sport\_club \sqsubseteq \forall\, has\text{-}sport\_club\text{-}name.name$$

Required and Optional Attributes. We use cardinality restrictions of respective object-properties on the owning *ENTITY* to model the attribute cardinality constraints in the ORA-SS model. The default is (0:1). We use a cardinality ≥ 1 restriction to state a required attribute.

Single-Valued vs. Multi-valued Attributes. Single-valued attributes can be modeled by specifying the respective object-property as functional. Multi-valued attributes, on the contrary, are not functional. An attribute is by default single valued.

Primary Key Attributes. For an entity with a primary key attribute, we use an allValuesFrom restriction on the property *has-primary-key* to constrain it. Since we have specified that *has-primary-key* is inverse functional, this suffices to show that two different objects will have different primary keys. Moreover, for every attribute that is the primary key attribute, we assert that the corresponding object property is a sub property of *has-primary-key*.

Disjunctive Attributes. Similar to the treatment of disjunctive relationship types, we create a class as the *union* of a set of disjunctive attribute classes. Together with the cardinality ≤ 1 restriction, disjunctive attributes can be represented in OWL.

3.5 Instance Diagrams in OWL

The representation of ORA-SS instance diagrams in OWL is a straightforward task. As the name suggests, instance diagrams are semistructured data instances of a particular ORA-SS schema diagram. The translation of an instance diagram to an OWL ontology is done by the following 3 steps:

1. Defining individuals and stating the membership of these individuals, by declaring them as instances of the respective OWL classes of object classes, relationship types and attributes defined in the schema diagram ontology.
2. For each OWL class, we state that all its instances are different from each other.
3. By making use of the object-properties defined in the schema diagram ontology, we state the relationships among the individuals.

4 Reasoning About ORA-SS Instance Models

In this section, we demonstrate the validation of ORA-SS schema and instance diagrams using OWL and RACER. We will again use Fig. 1 as the running example.

4.1 Validation of Schema Diagram Ontologies

In order to ensure the correctness of an ORA-SS schema diagram, a number of properties have to be checked, such as:

– The parent of a relationship type should be either a relationship type or an object class, where the child should only be an object class.
– The parent of a higher-degree relationship type (higher than 2) must be a relationship type.
– An object class or relationship type can have at most one primary key, which must be part of the candidate keys.

To manually check the validity of a given schema diagram against these constraints is a highly laborious and error-prone task. By following the methodology presented in this section systematically, a lot of potential violation of the above constraints can be

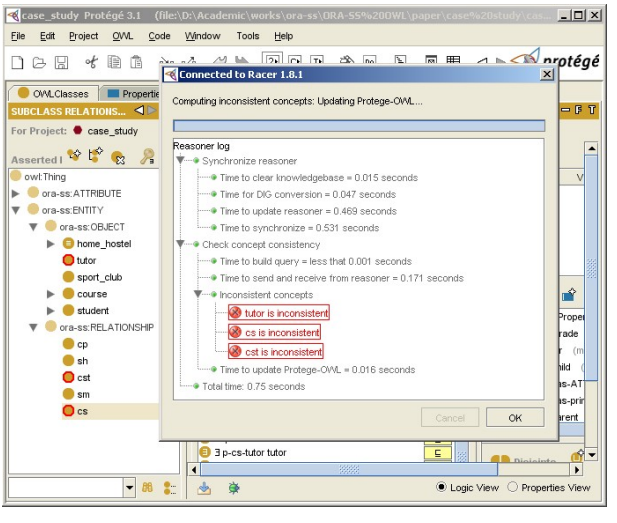

Fig. 2. Schema inconsistency detected by RACER

avoided. Moreover, the highly efficient OWL reasoners such as RACER can check the consistency of ORA-SS schema diagrams in OWL fully automatically. For example, suppose that in the case study, the child of relationship type *cs* is mistakenly put as *cst* instead of *student*_1. Hence, the axiom $\top \sqsubseteq \forall child.OBJECT$ is violated. This error can be picked up by RACER automatically, as shown in Fig. 2. Three classes, *cs*, *cst* and *tutor* are highlighted as inconsistent. Classes *cst* and *tutor* are inconsistent because they are both related to *cs* using existential or cardinality restrictions. Other types of checking can be similarly performed.

It can be seen from Fig. 2 that the detection of inconsistencies in the ORA-SS schema ontology by RACER is quite efficient. On a Pentium IV 2.4GHz machine with 1GB memory, the consistency checking by RACER took only 0.75 second.

4.2 Validation of Instance Diagram Ontologies

After transforming an ORA-SS instance diagram into an OWL ontology. Validation of the consistency of the instance ontology can be done fully automatically by invoking ontology reasoners capable of ABox reasoning. We will use RACER to demonstrate the checking of the above ontology using a few examples.

– Entity/attribute cardinality constraints
  In Fig. 1, each instance of relationship type *cst* has exactly one *tutor*. Suppose that in the instance ontology, $cs1$ is mapped to two tutors, $tutor1$ and $tutor2$ by *cst*.

$$\langle cs1, tutor1\rangle \in \textit{p-cs-tutor} \qquad \langle cs1, tutor2\rangle \in \textit{p-cs-tutor}$$

– Primary key related properties
  Suppose that by accident, two students, $student4$ and $student5$, are both assigned to the same student number.

$$\langle student4, student\_number\_4\rangle \in \textit{has-student-student\_number} \qquad \langle student5, student\_number\_4\rangle \in \textit{has-student-student\_number}$$

By using RACER And RacerPorter (a graphical front-end of RACER) together, the instance ontology is detected to be inconsistent automatically in the above two cases. In each case, RACER takes less than 1 second to conclude the incoherence of the ontology.

5 Conclusion

In this paper, we explored the synergy between the Semantic Web and the database modeling approaches in the context of verifying semistructured data design. We demonstrate the approach of using the OWL and its reasoning tool for the consistency checking of the ORA-SS data model and its instances. The advantages of our approach lie in the following perspectives. Firstly, we defined a Semantic Web ontology model for the ORA-SS data modeling language. It not only provides a formal semantic for the ORA-SS graphical notation, but also demonstrates that Semantic Web languages such as OWL can be used to capture more semantic information of a semistructured data. Furthermore, such a semantics can be adopted by many Semantic Web applications that use the

ORA-SS semistructured data model. Secondly, ontology reasoning tool was adopted to perform automated verification on a semistructured data model. The RACER reasoner was used to check the consistency of an ORA-SS schema model and its instances. We illustrated the various checking tasks through a `Course-Student` example model. In our previous work, we used the Alloy Analyzer for the validation of the ORA-SS data model. The main advantage of our current OWL approach over this is that consistency checking on large ORA-SS models are made feasible, as one of the shortcomings of the current Alloy Analyzer is its limited abilities on verifying large-scale models. Moreover, as Semantic Web reasoners employ highly optimized tableaux-based algorithms, the performance in terms of time is also significantly better than Alloy Analyzer.

References

1. V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. L. Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood. Document Object Model (DOM) Level 1 Specification. `http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/`.
2. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):35–43, 2001.
3. P. Buneman, S. B. Davidson, M. F. Fernandez, and D. Suciu. Adding Structure to Unstructured Data. In *ICDT '97: Proceedings of the 6th International Conference on Database Theory*, pages 336–350. Springer-Verlag, 1997.
4. G. Dobbie, X. Wu, T. Ling, and M. Lee. ORA-SS: Object-Relationship-Attribute Model for Semistructured Data. Technical Report TR 21/00, School of Computing, National University of Singapore, Singapore, 2001.
5. R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97: Proceedings of 23rd International Conference on Very Large Data Bases*, pages 436–445. Morgan Kaufmann, 1997.
6. V. Haarslev and R. Möller. Practical Reasoning in Racer with a Concrete Domain for Linear Inequations. In I. Horrocks and S. Tessaris, editors, *Proceedings of the International Workshop on Description Logics (DL-2002)*, Toulouse, France, Apr. 2002. CEUR-WS.
7. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.
8. T. Ling, M. Lee, and G. Dobbie. Applications of ORA-SS: An Object-Relationship-Attribute data model for Semistructured data. In *IIWAS '01: Proceedings of 3rd International Conference on Information Integration and Web-based Applications and Serives*, 2001.
9. T. W. Ling, M. L. Lee, and G. Dobbie. *Semistructured Database Design*. Springer, 2005.
10. 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.
11. X. Wu, T. W. Ling, M. L. Lee, and G. Dobbie. Designing Semistructured Databases Using the ORA-SS Model. In *WISE '01: Proceedings of 2nd International Conference on Web Information Systems Engineering*, Kyoto, Japan, 2001. IEEE Computer Society.