# Reasoning About ORA-SS Data Models Using the Semantic Web

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

[1] School of Computing, National University of Singapore, Republic of 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, The University of Manchester, United Kingdom
hwang@cs.manchester.ac.uk

**Abstract.** There has been a rapid growth in the use of semistructured data in both web applications and database systems. Consequently, the design of a good semistructured data model is essential. In the relational database community, algorithms have been defined to transform a relational schema from one normal form to a more suitable normal form. These algorithms have been shown to preserve certain semantics during the transformation. The work presented in this paper is the first step towards representing such algorithms for semistructured data, namely formally defining the semantics necessary for achieving this goal. 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 the ORA-SS notation into the Web Ontology Language (OWL) and automatically verifying the semistructured data design using the OWL reasoning tools. Our methodology provides automated consistency checking of an ORA-SS data model at both the schema and instance levels.

**Keywords:** Semistructured Data, Semantic Web, Ontology Web Language, ORA-SS, Formal Verification.

## 1 Introduction

Semistructured data has become prevalent in both web applications and database systems. With the growth in its usage, questions have arisen about the effective storage and management of semistructured data. In the relational database community, algorithms have been defined to transform a relational schema from one norm form to a more suitable normal form. These algorithms have been shown to preserve certain semantics during the transformation. In order to verify the correctness of the similar transformations for semistructured data, we need to have a standard representation of schemas and the transformation operators that are used to transform schemas. This process can be achieved by describing a formal model for semistructured data schemas and verifying that instances of schemas conform to the schema model. Then the basic transformation operators can be formally defined on this schema model. In this paper we undertake the first step of the process defined above.

S. Spaccapietra (Ed.): Journal on Data Semantics VII, LNCS 4244, pp. 219–241, 2006.

Many data modeling languages [1–4] for semistructured data have been introduced to capture more detailed semantic information. The Object Relationship Attribute model for Semistructured data (ORA-SS) [5, 6] is a semantically enriched graphical notation for designing and representing semistructured data [6–9]. 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 are its ability to express the semantics that are necessary for designing effective storage and management algorithms, such as 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 essential, even crucial for semistructured data representation and management, but it is lacking in other existing semistructured data modeling notations.

Semistructured data also acts as a hinge technology between the data exchanged on the web and the data represented in a database system. 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 [10] approach proposed by the World Wide Web Consortium (W3C) attracts the most attention. It is regarded as the next generation of the web. The Web Ontology Language (OWL) [11] is an ontology language for the Semantic Web. It consists of three increasingly expressive sub-languages: OWL Lite, DL and Full. 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 [12].

The reason that we chose the OWL ontology language is because of the strong relationship between semistructured data and 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 meaning to the data on the web. The ORA-SS language is a semantically enriched data modeling notation for describing semistructured data. From the point of capturing more semantic information in content representation, OWL and ORA-SS are two approaches that fulfil the same goal, where the former is rooted in the web community and the latter has its basis in the database community. Thus it is natural to explore the synergy of the two approaches. We believe that semantic web and its reasoning tools can contribute greatly to the design and verification phases of ORA-SS data models.

In this paper, we propose a methodology to verify ORA-SS data design using OWL and its reasoner RACER. Fig. 1 shows the overall process of our approach. 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 enables us to represent customized ORA-SS data models and their instances in OWL. Secondly, ORA-SS schema and instance models are translated into their corresponding OWL ontologies. Finally, RACER is used to perform the automated verification of the ORA-SS ontologies.

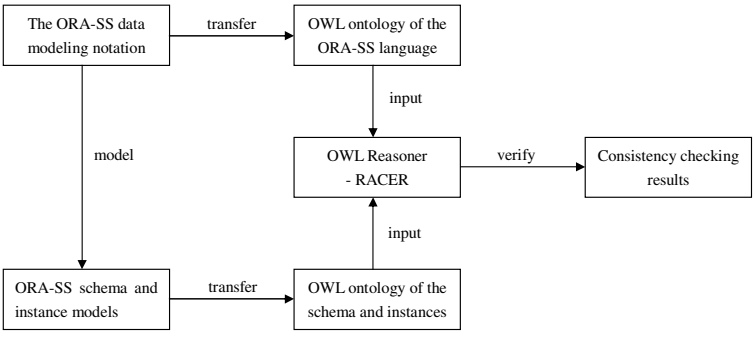


**Fig. 1.** The overall approach to verify ORA-SS data models using OWL

Our approach is able to provide automatic consistency checking on large ORA-SS data models and their instances. Examples are given through out the paper to illustrate the reasoning process.

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 semantics. Inconsistencies at the instance level arise if an ORA-SS instance model 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 objects 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 verifying ORA-SS semistructured data modeling is very beneficial.

There has been other research that provides a formal semantics for semistructured data. For example, the formalization of DTD (Document Type Definition) and XML declarative description documents using expressive description logic has been presented by Calvanese et al. [13]. Anutariya et al. presented the same formalization using a theoretical framework developed using declarative description theory [14]. Also spatial tree logics has been used to formalize semistructured data by Conforti and Ghelli [15]. More recently, hybrid multimodal logic was used to formalize semistructured data by Bidoit et al. [16]. We also applied a similar approach to formalize ORA-SS data models using Z/EVES [17]. While this work has helped us develop a better understanding of the semantics of semistructured data, it does not provide automated verification. In another research we presented a formalization of the ORA-SS notation in the Alloy [18] language. Although the automated verification was available using the Alloy Analyzer, it had a scalability problem, making the verification of large sets of semistructured data impossible. In addition, there were also research for providing better validation support of semistructured data, such as algorithms on incremental validation of XML documents [19, 20]. However, these approaches still focused on the syntax-only checking of semistructured data 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 the ontology reasoning process through a *Course-Student* ORA-SS data model example. Examples of both class-level reasoning and instance-level reasoning are presented. Section 5 concludes the paper and outlines the future work.

## 2 Background

### 2.1 The ORA-SS Data Modeling Language

The Object-Relationship-Attribute model for Semistructured data (ORA-SS) is a semantically enriched data modeling language for semistrctured data design [5, 6]. It has been used in many XML related database applications [8, 9]. The ORA-SS notation consists of four basic concepts: object class, relationship type, attribute and reference. A full description of the ORA-SS data modeling language can be found in [5, 6].

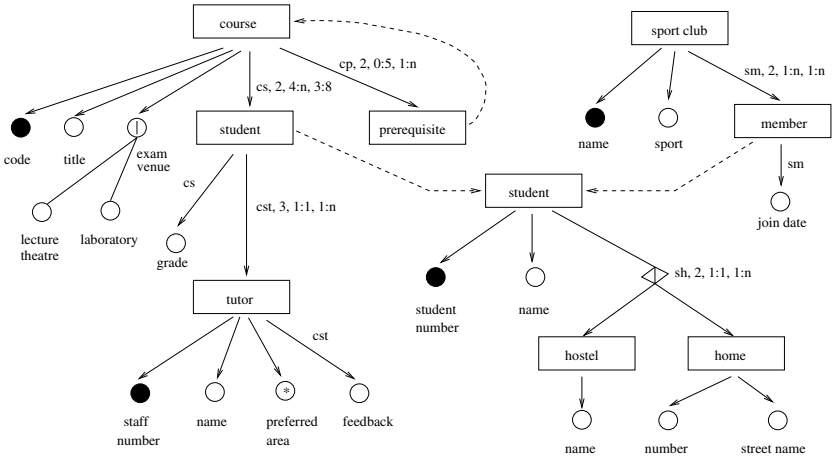


**Fig. 2.** The ORA-SS schema diagram of a *Course-Student* data model

- 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 is represented as a filled circle. Other types of attributes include single valued attribute, multi-valued attribute, required attribute, 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 represents the constraints on relationships, participations and cardinalities among the instances of the object classes in a semistructured data model. For example, Fig. 2 represents an ORA-SS schema diagram of a `Course-Student` data model. In the diagram, each `course` has `code`, `title`, and `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. This checking could be done manually only if it is a relatively small sized model. However, examining complicated and large ORA-SS data models for semistructured data is almost manually impossible. Furthermore, manual diagrammatic checking does not guarantee the consistency of the schema since it is likely that inconsistencies are not revealed when the schema is large and complicated. Therefore, automated verification support based on the formal specification of ORA-SS semantics is highly desirable.

### 2.2 Semantic Web – OWL and RACER

Description logics [21] 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. Based on RDF Schema [22] and DAML+OIL [23], the Web Ontology Language (OWL) [11] is the de facto ontology language for the Semantic Web. It consists of three increasingly expressive sub-languages: 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 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. Table 1 summarizes the ‘DL syntax’ used in the following sections. Interested readers may refer to [11] for full details.

RACER, the **R**enamed **A**Box and **C**oncept **E**xpression **R**easoner [12], is a reasoning engine for ontology languages DAML+OIL and OWL. It implements a TBox and ABox

**Table 1.** Summary of OWL syntax used in the paper

| Notation | Explanation |
|---|---|
| $\top / \bot$ | Super class/sub class of every class |
| $N_1 \sqsubseteq N_2$ | $N_1$ is a sub class/property of $N_2$ |
| $C_1 = C_2$ | Class equivalence |
| $C_1 \sqcap / \sqcup C_2$ | Class intersection/union |
| $\geq 1\ P \sqsubseteq C$ | Domain of property $P$ is class $C$ |
| $\top \sqsubseteq \forall P.C$ | Range of $P$ is $C$ |
| $\top \sqsubseteq \leq 1\ P$ | Property $P$ is functional |
| $P_2 = (^{-}P_1)$ | Property $P_2$ is inverse of $P_1$ |
| $\forall / \exists P.C$ | allValuesFrom/someValuesFrom restriction, giving the class that for every instance of this class that has instances of property $P$, the values of the property are all/some members of the class $C$ |
| $= / \leq / \geq n\ P$ | Cardinality restriction, the class each of whose instances mapped by property $P$ forms a set whose cardinality must be exactly/less than/greater than $n$ |

reasoner for the description logic $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D})^-$ [12]. It is automated for reasoning over OWL Lite and DL ontologies.

## 3 Modeling the ORA-SS Data Model in OWL

In this section, we present the modeling of ORA-SS schema and instance diagram as OWL ontologies in three steps. Firstly, we define the ORA-SS ontology, which contains the OWL definitions of essential ORA-SS concepts, such as object class, relationship type, etc. Secondly, we show how individual ORA-SS schema diagram ontologies can be constructed based on the ORA-SS ontology together with a case tool for achieving this. Finally, in Section 3.6, we show how ORA-SS instance diagrams can be represented in OWL. To effectively illustrate the modeling approach, the schema diagram in Fig. 2 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 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

[1] Due to the space limit, only part of the ORA-SS OWL semantics are presented in the paper. A complete ORA-SS ontology can be found at `http://www.comp.nus.edu.sg/~liyf/ora-ss/ora-ss.owl`.

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 seem more 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, when describing the relationships of ternary and more. Secondly, classes and properties in OWL DL are disjoint. In our model, an OWL relationship type 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 for 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.

1. **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.
   *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.

   $\geq 1\ parent \sqsubseteq RELATIONSHIP$ $\quad$ $\geq 1\ child \sqsubseteq RELATIONSHIP$
   $\top \sqsubseteq \forall\, parent.ENTITY$ $\quad$ $\top \sqsubseteq \forall\, child.OBJECT$
   $\top \sqsubseteq \leq 1\ parent$ $\quad$ $\top \sqsubseteq \leq 1\ child$

   $OBJECT \sqsubseteq \neg\ \exists\, parent.\top$ $\quad$ $RELATIONSHIP \sqsubseteq \forall\, parent.ENTITY$
   $OBJECT \sqsubseteq \neg\ \exists\, child.\top$ $\quad$ $RELATIONSHIP \sqsubseteq \forall\, child.OBJECT$

   *Secondly*, we define two more object-properties: *p-ENTITY-OBJECT* and *p-OBJECT-ENTITY*. These two properties are the 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 inverses, the domain and range of *p-OBJECT-ENTITY* can be deduced.

$p\text{-}OBJECT\text{-}ENTITY = (^{-}p\text{-}ENTITY\text{-}OBJECT)$
$\geq 1\ p\text{-}ENTITY\text{-}OBJECT \sqsubseteq ENTITY$ $\quad\quad \geq 1\ p\text{-}OBJECT\text{-}ENTITY \sqsubseteq OBJECT$
$\top \sqsubseteq \forall\, p\text{-}ENTITY\text{-}OBJECT.OBJECT$ $\quad\quad \top \sqsubseteq \forall\, p\text{-}OBJECT\text{-}ENTITY.ENTITY$

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

2. **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*.

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

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*.

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

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

### 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. 2 contains a number of object classes [2].

$course \sqsubseteq OBJECT$ $\quad\quad tutor \sqsubseteq OBJECT$
$student \sqsubseteq OBJECT$ $\quad\quad sport\_club \sqsubseteq OBJECT$
$hostel \sqsubseteq OBJECT$ $\quad\quad home \sqsubseteq OBJECT$
$\ldots$ $\quad\quad \ldots$

**Referencing** – In ORA-SS, an object class can reference another object class to refer to its definition. We say that a *reference* object class references a *referenced* object class. In our model, we model the *reference* object class as a sub-class of the *referenced* object class. If the two object classes have 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.

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

*Example 2.* In Fig. 2, 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.

$$student \sqsubseteq OBJECT \qquad student\_1 \sqsubseteq student \qquad member \sqsubseteq student$$

### 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.

*Example 3.* Fig. 2 contains 5 relationship types.

$$cs \sqsubseteq RELATIONSHIP \qquad sm \sqsubseteq RELATIONSHIP \qquad cst \sqsubseteq RELATIONSHIP$$
$$sh \sqsubseteq RELATIONSHIP \qquad cp \sqsubseteq RELATIONSHIP$$

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*.

$$cs \sqsubseteq \forall parent.course \qquad cs \sqsubseteq \forall child.student\_1$$
$$cs \sqsubseteq \exists parent.course \qquad cs \sqsubseteq \exists child.student\_1$$

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

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

$$\geq 1\ p\text{-}course\text{-}student \sqsubseteq course \qquad \geq 1\ p\text{-}student\text{-}course \sqsubseteq student\_1$$
$$\top \sqsubseteq \forall p\text{-}course\text{-}student.student\_1 \qquad \top \sqsubseteq \forall p\text{-}student\text{-}course.course$$

**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 5.* In Fig. 2, 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 relationship consists of disjunctive object classes, where only one object can be selected to the relationship instance from the set of disjunctive 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 6.* In Fig. 2, *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*.

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

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

$$student \sqsubseteq \forall p\text{-}student\text{-}sh.hostel\_home \qquad student \sqsubseteq = 1\ p\text{-}student\text{-}sh$$

**Recursive Relationship Types** – Recursive relationship types in ORA-SS are modeled using referencing. In our model, by defining the reference object class as a sub class of the referenced object class, recursive relationship types can be modeled as a regular relationship type.

*Example 7.* Fig. 2 depicts such a recursive relationship type where a *course* object class has at most 5 *prerequisite* objects, whereas a *prerequisite* must have at least one *course*. This can be modeled as follows.

$$course \sqsubseteq \top \qquad prerequisite \sqsubseteq course$$
$$\geq 1\ p\text{-}course\text{-}prerequisite \sqsubseteq course \qquad \top \sqsubseteq \forall p\text{-}course\text{-}prerequisite.prerequisite$$
$$p\text{-}prerequisite\text{-}course = (^{-}p\text{-}course\text{-}prerequisite)$$

$$course \sqsubseteq \forall p\text{-}course\text{-}prerequisite.prerequisite \qquad prerequisite \sqsubseteq \forall p\text{-}prerequisite\text{-}course.course$$
$$course \sqsubseteq \leq 5\ p\text{-}course\text{-}prerequisite \qquad prerequisite \sqsubseteq \geq 1\ p\text{-}prerequisite\text{-}course$$

### 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 8.* The schema diagram in Fig. 2 generates the following OWL classes for attributes.

$code \sqsubseteq ATTRIBUTE$ $grade \sqsubseteq ATTRIBUTE$
$title \sqsubseteq ATTRIBUTE$ $student\_number \sqsubseteq ATTRIBUTE$
$exam\_venue \sqsubseteq ATTRIBUTE$ $sport \sqsubseteq ATTRIBUTE$
...

**Modeling Various Definitions** – As OWL adopts the Open World Assumption [11] 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 *conventions*.

– For each *ENTITY*, we use an *allValuesFrom* restriction on *has-ATTRIBUTE* over the union of all its *ATTRIBUTE* classes. This denotes the complete set of attributes that the *ENTITY* holds.

*Example 9.* 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. For an entity *Ent* and its attribute *Att*, we have the following definitions.

$has\text{-}Ent\text{-}Att \sqsubseteq has\text{-}ATTRIBUTE$ $\quad \top \sqsubseteq \forall\, has\text{-}Ent\text{-}Att.Att$
$\geq 1\ has\text{-}Ent\text{-}Att \sqsubseteq Ent$

*Example 10.* In Fig. 2, the object class *sport club* has an attribute name. It can be modeled as follows.

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

**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 $\geq 1$ restriction to state a required attribute.

*Example 11.* Take sport club as an example again, it can have 0 or 1 sport.

$$sport\_club \sqsubseteq \leq 1\ has\text{-}sport\_club\text{-}sport$$

**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.

*Example 12.* In Fig. 2, object tutor has a single-valued attribute name. This can be modeled as follow.

$\geq 1\ has\text{-}tutor\text{-}name \sqsubseteq tutor$ $\quad has\text{-}tutor\text{-}name.name \sqsubseteq has\text{-}ATTRIBUTE$
$\top \sqsubseteq \forall\, has\text{-}tutor\text{-}name.name$ $\quad \top \sqsubseteq \leq 1\ has\text{-}tutor\text{-}name$

**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*.

*Example 13.* In Fig. 2, object class *course* has an attribute *code* as its primary key and this is modeled as follows. The `hasValuesFrom` restriction enforces that each individual must have some *code* value as its primary key.

$$course \sqsubseteq \forall\, has\text{-}primary\text{-}key.code \qquad course \sqsubseteq \exists\, has\text{-}primary\text{-}key.code$$

**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 $\leq 1$ restriction, disjunctive attributes can be represented in OWL.

*Example 14.* In Fig. 2, *course* has a disjunctive attribute exam venue, which is either lecture theater or laboratory. It can be modeled as follows.

$$lecture\_theatre \sqsubseteq ATTRIBUTE \qquad lecture\_theatre \sqcap laboratory = \bot$$
$$laboratory \sqsubseteq ATTRIBUTE \qquad exam\_venue = lecture\_theatre \sqcup laboratory$$
$$exam\_venue \sqsubseteq ATTRIBUTE$$

$$course \sqsubseteq \forall\, \text{has-course-exam\_venue}.exam\_venue$$
$$course \sqsubseteq \leq 1\ \text{has-course-exam\_venue}$$

**Fixed-Value Attributes** – A fixed-value attribute is one whose value is the same for every instance and cannot be changed. To model this, we define the attribute to be an OWL class that has only one instance. Suppose that the object *obj* has a fixed-value attribute *attr*, whose value is *attr_val*. The OWL ontology will then contain the following statements.

$$obj \sqsubseteq OBJECT \qquad has\text{-}obj\text{-}attr \sqsubseteq has\text{-}ATTRIBUTE$$
$$attr \sqsubseteq ATTRIBUTE \qquad \geq 1\ has\text{-}obj\text{-}attr \sqsubseteq obj$$
$$attr\_val \in attr \qquad \top \sqsubseteq \forall\, has\text{-}obj\text{-}attr.attr$$
$$attr = \{attr\_val\}$$

### 3.5 Presenting and Transforming ORA-SS Diagrams in OWL

In the previous subsections, we presented some of the formal definitions of the ORA-SS language constructs in OWL. Part of the ontology (in OWL XML syntax) of the ORA-SS schema diagram in Fig. 2 is shown below.

```
<owl:Class rdf:about="#student">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.comp.nus.edu.sg/
                    ~liyf/ora-ss/ora-ss.owl#has-primary-key"/>
```

```
      <owl:someValuesFrom rdf:resource="#student_number"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:maxCardinality rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#int">1
      </owl:maxCardinality>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID=
                    "has-student-student_number"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
```

Note that because OWL has XML syntax as its presentation form, further automated transformation tools can be easily developed to assist the translation from ORA-SS data models into their corresponding OWL representations. We are in the process of developing a visual case tool which provides a high-level and intuitive environment for constructing ORA-SS data models in OWL. Our ORA-SS modeling tool was built based on the meta-tool Pounamu [24]. Pounamu is a meta-case tool for developing multi-view visual environments. Fig. 3 shows the *Course-Student* schema example in section 2.1 defined by the tool. From the diagram, we can see that the customized schema model

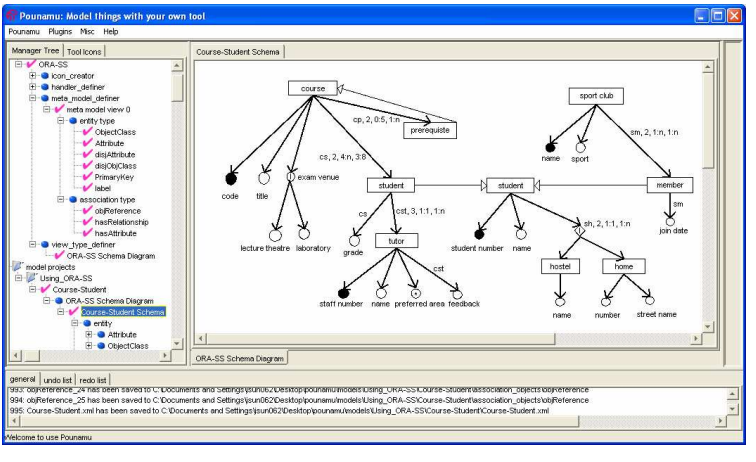


**Fig. 3.** A case tool for ORA-SS data modeling

can be defined easily by creating instances of the pre-defined model entities and associations. By triggering the defined event handler menu item in the tool, it transforms all the default XML format of each entity in the diagram into a single OWL representation of the ORA-SS schmea model and saves it for later reasoning purpose. One ongoing development is to develop our tool as a plugin within the overall Protégé plug-and-play framework.

In the next section, we show how to model ORA-SS instance diagrams using OWL individuals based on the classes, properties defined in the OWL schema ontology.

### 3.6 Instance Diagrams in OWL

The representation of ORA-SS instance diagrams [3] 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 described 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.

This is best illustrated with an example. We create an instance ontology for the schema ontology defined in Fig. 2. In this paper, we use a table form to illustrate the ORA-SS OWL instances. This is just for the sake of easy representation. Actual ORA-SS instances are defined in the ORA-SS instance diagrams and transformed into their corresponding OWL representations in XML.

– In Table 2 below, we give a brief overview of the individuals under respective object classes [4].

**Table 2.** Instances of various objects of Fig. 2

| Object classes | Instances |
|---|---|
| course | course1, course2, course3, course4 |
| student | student1, student 2, ..., student8 |
| prerequisite | course1, course2, course3, course4 |
| home | home1, home2 |
| hostel | hostel1, hostel2, hostel3 |
| tutor | tutor1, tutor2, tutor3 |
| sport club | club1, club2, club3 |
| member | student1, ..., student6 |

– Next, we define the instances of various attributes of Fig. 2 in Table 3 listed in the appendix.
– Having defined all the instances of objects and attributes, the next step is to relate them. We proceed by populating various memberships. In Table 4 listed in the appendix, we show the pairs of instances related by each relationship type in Fig. 2. It is worthwhile pointing out that ternary or higher-degree relationships are viewed

[3] In this paper, we present the schema instances in a table format rather than in a proper ORA-SS instance diagram. This is just for the sack of presentation purpose only.
[4] Due to the space limit, not all the OWL individuals of the schema diagram in Fig. 2 are shown.

as pairs of pairs. Also note that for brevity reasons, we will refer to the members of relationships such as *cs*, *cst* and *sm* as *cs1,.., cs24*, *cst1,.. cst24* and *sm1, .. sm6* respectively.

– The last task in modeling this instance diagram in OWL is to associate object classes and relationship types to the attributes. We show the instances of object classes and relationship types of Fig. 2 in Table 5 listed in the appendix. Note that attributes whose names are in italic and bold fonts are primary key attributes.

By following the above steps, we can easily represent an ORA-SS instance diagram in OWL [5]. Similarly, an OWL instance diagram generation functionality is under development in the ORA-SS case tool presented earlier. In addition, direct transformation from an XML document into its ORA-SS OWL instance can also be implemented. With the constraints defined in the OWL schema ontology, we are able to perform automated reasoning over these instances (OWL individuals), as detailed in the next section.

## 4 Reasoning About ORA-SS Data Models

In this section, we demonstrate the validation of ORA-SS schema and instance diagrams using OWL and RACER. We will again use Fig. 2 as the running example.

### 4.1 Verification 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.
– The child participants of a disjunctive relationship type or attribute must be a set of disjunctive object classes or attributes.
– A composite attribute or disjunctive attribute has an attribute that is related to two or more sub-attributes.
– A candidate (primary) key attribute of an object class must be selected from the set of attributes of the object class.
– A composite key is selected from 2 or more attributes of an object class.
– An object class or relationship type can have at most one primary key, which must be part of the candidate keys.
– Relationship attributes have to relate to an existing relationship type.
– An object class can reference one object class only, but an object class can be referenced by multiple object classes.

[5] The complete OWL representation of the *Course-Student* instance example can be found at`http://www.comp.nus.edu.sg/~liyf/ora-ss/case_instance.owl`.

The above are some of the criteria for validating a schema diagram against the ORA-SS notation. 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, potential violations of the above constraints can be avoided. This is because the formal semantics of the OWL language allows precise specifications to be expressed.

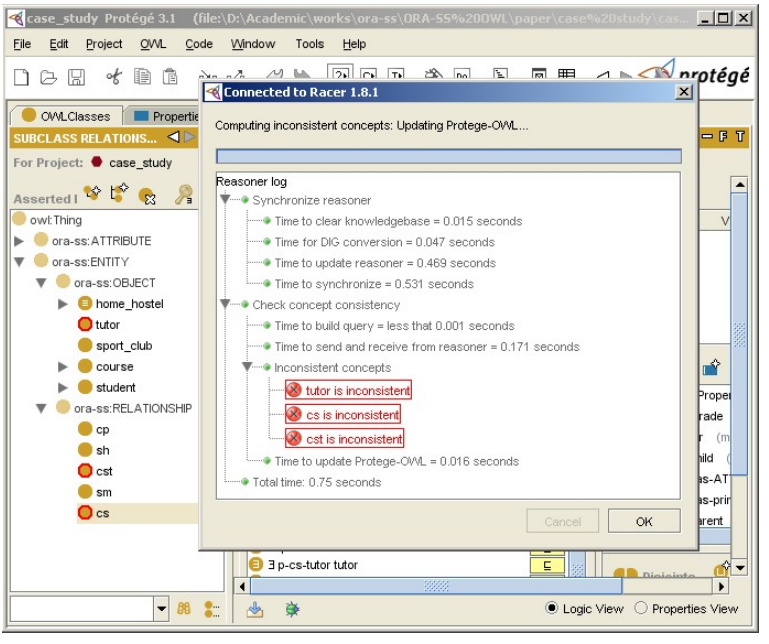


**Fig. 4.** Schema inconsistency detected by RACER

Moreover, highly efficient OWL reasoners such as RACER can check the consistency of ORA-SS schema diagrams in OWL automatically. For example, suppose that in Fig. 2, the child of relationship type *cs* is mistakenly associated with a relationship type *cst* instead of the reference object class *student_1*. This error can be picked up by RACER automatically, as shown in Fig. 4. Three classes, *cs*, *cst* and *tutor* are highlighted as inconsistent. It is inconsistent because both *cst* and *tutor* are related to *cs* using existential or cardinality restrictions. Other types of checking can be similarly performed.

### 4.2 Verification of Instance Diagram Ontologies

The ORA-SS instance validation is defined to check whether there are any possible inconsistencies in a semistructured data instance, where an ORA-SS instance should be consistent with regard to the designated ORA-SS schema diagram. Possible guidelines for validating an ORA-SS instance are as follow.

- Relationship instances must conform to the parent participation constraints, e.g., the number of child objects related to a single parent object or relationship instance should be consistent with the parent participation constraints; and the number of parent objects or relationship instances that a single child object relates to should be consistent with the child participation constraints.
- In a disjunctive relationship, only one object class can be selected from the disjunctive object class set and associated to a particular parent instance.
- For a candidate key (single or composite), its value should uniquely identify the object that this key attribute belongs to.
- Each object can have one and only one primary key.
- All attributes have their own cardinality and the number of attributes that belong to an object should be limited by the minimum and maximum cardinality values of the attribute.
- For a set of disjunctive attributes, only one of the attribute choices can be selected and associated to an object instance.

These are some of the criteria of instance level validation. Given an ORA-SS instance, we first transform it into its corresponding OWL instance representation, then verify the consistency of the instance ontology automatically by invoking ontology reasoners capable of ABox reasoning. We will use RACER to demonstrate the checking of the above ontology through a few examples.

- Entity/attribute cardinality constraints
  In the schema ontology, each instance of relationship type *cst* has exactly one *tutor*. Suppose that in the instance ontology, *(course1, student)* has both *tutor1* and *tutor2* as the child for the relationship type *cst*.

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

  By using RACER, the instance ontology is detected to be inconsistent.
- Primary key related properties
  Suppose that by accident, two students, *student4* and *student5*, are both assigned to the same student number.

  $$\langle student4, student\_number\_4\rangle \in student\_number \qquad \langle student5, student\_number\_4\rangle \in student\_number$$

  Similarly, RACER instantly detects the inconsistency.

### 4.3 Debugging ORA-SS OWL Models

The OWL reasoners, such as RACER, can perform efficient reasoning on large ontologies automatically. They can detect whether an OWL ontology is consistent or not *as a whole*. However, it is not capable of locating which individual caused the inconsistency. When checking satisfiability (consistency), the OWL reasoners can only provide a list of unsatisfiable classes and offer no further explanation for their unsatisfiability. It means that the reasoner can only conclude if an ORA-SS data model is consistent and flag the invalid class or individuals. The process of 'debugging' an ORA-SS data

model is left for the user. When faced with several unsatisfiable individuals in a moderately large ORA-SS data model, even expert ontology engineers can find it difficult to work out the underlying error. Debugging an ontology has been well recognized as a non-trivial task. To provide some debugging assistance for the inconsistent ORA-SS models, we have built an OWL debugging tool [6] based on the heuristics [25]. Our OWL debugger has been designed to adopt the general OWL DL ontology and it can also be used to explain the errors in the ORA-SS data models as well.

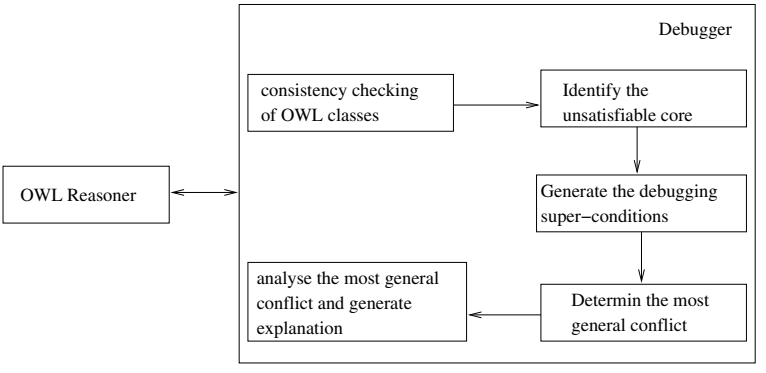


**Fig. 5.** The debugging process

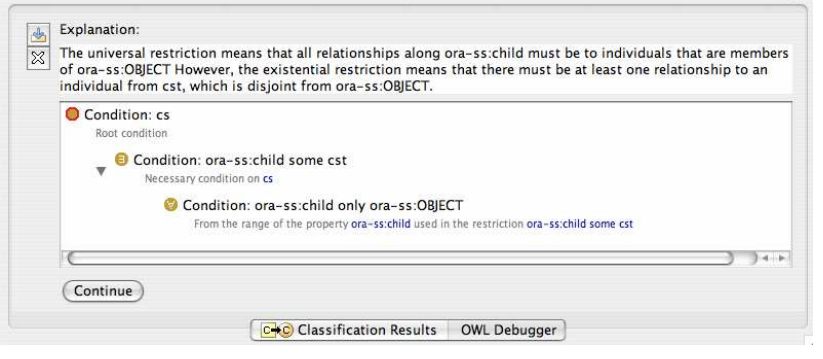


**Fig. 6.** Debugging the ORA-SS schema model

Figure 5 illustrates the main steps of the debugging process. The user selects an OWL class for debugging, which is checked to ensure it is indeed inconsistent, and that the user is making a valid request to the debugger. The debugger then attempts to identify the *unsatisfiable core* for the input class in order to minimize the search space. The *unsatisfiable core* is the smallest set of local conditions (direct super classes) that

[6] The work is supported in part by the CO-ODE project funded by the UK Joint Information Services Committee and the HyOntUse Project (GR/S44686) funded by the UK Engineering and Physical Science Research Council.

leads to the class in question being inconsistent. Having determined the unsatisfiable core, the debugger attempts to generate the *debugging super conditions*, which are the conditions that are implied by the conditions in the *unsatisfiable core*. The debugger then examines the *debugging super conditions* in order to identify the *most general conflicting* class set, which is analyzed to produce an explanation as to why the class in question is inconsistent.

For example, Figure 6 shows the result of the debugging for the inconsistent example in Section 4.1, where the child of relationship type *cs* is mistakenly associated with another relationship type *cst* instead of the reference object class *student_1*. Previously, three classes, *cs*, *cst* and *tutor* were highlighted as inconsistent by the RACER. With the help of our debugger, it was pinpointed that the *cs* should have an *OBJECT* in its *child* relationship and not an individual from the *cst* relationship type. This is exactly the reason for the inconsistency as found using manual inspection earlier.

## 5 Conclusion

In this paper, we explored the synergy between the Semantic Web and the database modeling approaches in the context of validating semistructured data design. We demonstrate the approach of using OWL and its reasoning tool for consistency checking of the ORA-SS data models and their instances. The advantages of our approach lie in the following aspects. 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 deeper semantic information of semistructured data. Furthermore, such semantics can be adopted by many Semantic Web applications that use the ORA-SS semistructured data model. Secondly, an ontology reasoning tool was adopted to perform automated verification of ORA-SS data models. 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 [18]. The main advantage of our OWL approach over this is that consistency checking on large ORA-SS data models are made feasible. The current OWL reasoner can classify an ontology with 30,000 concepts within 4 seconds [26], which well satisfies the needs for any real-sized ORA-SS schemas and their instances.

In the future, we will further develop the visual case tool for editing and autogeneration of ORA-SS data models into their corresponding OWL representations for machine verification. Furthermore, we plan to extend and concentrate our work on defining the basic transformation operators that are used to transform ORA-SS schemas and providing verification for transformed schemas of semistructured data. By doing so, verifying the results of applications or databases that transforms the schema of semistructured data can be possible. In addition, we plan to extend the semantics of ORA-SS in OWL to investigate normalization issues in semistructured data design. The normal form of the ORA-SS data model for designing semistructured databases has been proposed in [9]. We would like to verify whether the semantics of a normalized schema is the same as its original form, showing whether a normalization algorithm changes the semantics of the schema during the transformation process.

## References


1. Apparao, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Hors, A.L., Nicol, G., Robie, J., Sutor, R., Wilson, C., Wood, L.: Document Object Model (DOM) Level 1 Specification (1998) http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/.
2. Buneman, P., Davidson, S.B., Fernandez, M.F., Suciu, D.: Adding Structure to Unstructured Data. In: ICDT '97: Proceedings of the 6th International Conference on Database Theory, Springer-Verlag (1997) 336–350
3. Goldman, R., Widom, J.: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In Jarke, M., Carey, M.J., Dittrich, K.R., Lochovsky, F.H., Loucopoulos, P., Jeusfeld, M.A., eds.: VLDB'97: Proceedings of 23rd International Conference on Very Large Data Bases, Morgan Kaufmann (1997) 436–445
4. McHugh, J., Abiteboul, S., Goldman, R., Quass, D., Widom, J.: Lore: A Database Management System for Semistructured Data. SIGMOD Record **26**(3) (1997) 54–66
5. Dobbie, G., Wu, X., Ling, T., Lee, M.: ORA-SS: Object-Relationship-Attribute Model for Semistructured Data. Technical Report TR 21/00, School of Computing, National University of Singapore, Singapore (2001)
6. Ling, T.W., Lee, M.L., Dobbie, G.: Semistructured Database Design. Springer (2005)
7. Chen, Y., Ling, T.W., Lee, M.L.: A Case Tool for Designing XML Views. In: DIWeb'02: Proceedings of the 2nd International Workshop on Data Integratino over the Web, Toronto, Canada (2002) 47–57
8. Ling, T., Lee, M., Dobbie, G.: 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. Wu, X., Ling, T.W., Lee, M.L., Dobbie, G.: Designing Semistructured Databases Using the ORA-SS Model. In: WISE '01: Proceedings of 2nd International Conference on Web Information Systems Engineering, Kyoto, Japan, IEEE Computer Society (2001)
10. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284**(5) (2001) 35–43
11. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. J. of Web Semantics **1**(1) (2003) 7–26
12. Haarslev, V., Möller, R.: Practical Reasoning in Racer with a Concrete Domain for Linear Inequations. In Horrocks, I., Tessaris, S., eds.: Proceedings of the International Workshop on Description Logics (DL-2002), Toulouse, France, CEUR-WS (2002)
13. Calvanese, D., Giacomo, G.D., Lenzerini, M.: Representing and Reasoning on XML Documents: A Description Logic Approach. Journal of Logic and Computation **9**(3) (1999) 295–318
14. Anutariya, C., Wuwongse, V., Nantajeewarawat, E., Akama, K.: Towards a Foundation for XML Document Databases. In: EC-Web. (2000) 324–333
15. Conforti, G., Ghelli, G.: Spatial Tree Logics to reason about Semistructured Data. In: SEBD. (2003) 37–48
16. Bidoit, N., Cerrito, S., Thion, V.: A First Step towards Modeling Semistructured Data in Hybrid Multimodal Logic. Journal of Applied Non-Classical Logics **14**(4) (2004) 447–475
17. Lee, S.U., Sun, J., Dobbie, G., Li, Y.F.: A Z Approach in Validating ORA-SS Data Models. In: 3rd International Workshop on Software Verification and Validation, Electronic Notes in Theoretical Computer Science, Volume 157, Issue 1, Manchester, United Kingdom (2005) 95-109
18. Wang, L., Dobbie, G., Sun, J., Groves, L.: Validating ORA-SS Data Models using Alloy. In: The Australian Software Engineering Conference (ASWEC 2006), Sydney, Australia (2006) 231–240

19. Papakonstantinou, Y., Vianu, V.: Incremental validation of XML documents. In: 9th International Conference on Database Theory (ICDT 2003), Siena, Italy, Springer (2003) 47–63
20. Bouchou, B., Alves, M.H.F.: Updates and Incremental Validation of XML Documents. In: Database Programming Languages: 9th International Workshop, DBPL 2003, Potsdam, Germany, Springer (2003) 216–232
21. Nardi, D., Brachman, R.J.: An introduction to description logics. In Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The description logic handbook: theory, implementation, and applications. Cambridge University Press (2003) 1–40
22. D. Brickley and R.V. Guha (editors): Resource description framework (rdf) schema specification 1.0. `http://www.w3.org/TR/rdf-schema/` (2004)
23. Harmelen, F., Patel-Schneider, P.F., (editors), I.H.: Reference description of the DAML+OIL ontology markup language. Contributors: T. Berners-Lee, D. Brickley, D. Connolly, M. Dean, S. Decker, P. Hayes, J. Heflin, J. Hendler, O. Lassila, D. McGuinness, L. A. Stein, et. al. (March, 2001)
24. Zhu, N., Grundy, J., Hosking, J.: Pounamu: a meta-tool for multi-view visual language environment construction. In: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'04), Rome, Italy (2004)
25. Wang, H., Horridge, M., Rector, A., Drummond, N., Seidenberg, J.: Debugging OWL-DL Ontologies: A Heuristic Approach. In: Proc. of 4th International Semantic Web Conference (ISWC'05), Galway, Ireland, Springer-Verlag (2005)
26. Tsarkov, D., Horrocks, I.: Optimised Classification for Taxonomic Knowledge Bases. In: Proceedings of the 2005 International Workshop on Description Logics (DL-2005), Edinburgh, United Kingdom (2005)

## Appendix

**Table 3.** Instances of various attributes of the ORA-SS schema diagram in Fig. 2

| **Attribute** | **Instances** |
|---|---|
| code | CS1101, CS1301, MA1102, CS2104 |
| title | "Java Programming", "Computer Architecture", "Calculus", "Programming Languages" |
| lecture threater | LT27, LT34, LT8 |
| laboratory | SR6, PL1, PL2 |
| grade | A, B, C, D, F |
| student number | stu no 1, stu no2, stu no3, ..., stu no 8 |
| name (student) | Jim, Gill, Mike, Rudy, Martin, Shirley, Tracy, Keith |
| number (home) | 20-22, 6-7 |
| street name (home) | Sunset Avenue, Avenue George V |
| name (hostel) | KR, SH, KEVII |
| staff number | stf no 1, stf no 2, stf no 3 |
| name (staff) | J-Sun, G-Dobbie, M-Jacksoon |
| preferred area | RE, SE, FM, DB, Network, Grid, SW |
| feedback | positive, negative, neutral |
| name (sport club) | yachting club, boxing club, tennis club |
| sport | yachting, boxing, tennis |
| join date | aug-02-2002, jan-25-2003, may-15-2003, oct-01-2003, dec-31-2004, jul-19-2005 |

**Table 4.** Instances of relationship types of the ORA-SS schema diagram in Fig. 2

| **Relationship types** | **Members** |
|---|---|
| cs | (course1, student1), (course1, student2), (course1, student3), (course1, student4), (course1, student5), (course1, student6), (course1, student7), (course1, student8), (course2, student5), (course2, student6), (course2, student7), (course2, student8), (course2, student1), (course2, student2), (course2, student3), (course2, student4), (course3, student1), (course3, student3), (course3, student5), (course3, student7), (course4, student2), (course4, student4), (course4, student6), (course4, student8) |
| cp | (course4, course1) |
| sh | (student1, home1), (student2, home2), (student3, hostel1), (student4, hostel2), (student5, hostel3), (student6, home1), (student7, hostel), (student8, hostel2) |
| cst | ((course1, student1), tutor1), ((course1, student2), tutor2), ((course1, student3), tutor3), ((course1, student4), tutor1), ((course1, student5), tutor2) ((course1, student6), tutor3), ((course1, student7), tutor1), ((course1, student8), tutor2), ((course2, student5), tutor3), ((course2, student6), tutor1), ((course2, student7), tutor2), ((course2, student8), tutor3), ((course2, student1), tutor1), ((course2, student2), tutor2), ((course2, student3), tutor3), ((course2, student4), tutor1), ((course3, student1), tutor2), ((course3, student3), tutor3), ((course3, student5), tutor1), ((course3, student7), tutor2), ((course4, student2), tutor3), ((course4, student4), tutor1), ((course4, student6), tutor2), ((course4, student8), tutor3) |
| sm | (club1, student1), (club1, student3), (club2, student2), (club2, student4), (club3, student5), (club3, student6) |

**Table 5.** Attribute values associated with the objects and relationships in Fig. 2

| **Entity** | **Attribute association** | | | |
|---|---|---|---|---|
| course | *instances* | ***code*** | *title* | *exam venue* |
| | course1 | CS1101 | "Java Programming" | LT27 |
| | course2 | CS1301 | "Computer Architecture" | LT34 |
| | course3 | MA1102 | "Calculus" | LT8 |
| | course4 | CS2104 | "Programming Languages" | SR6 |
| student | *instances* | *name* | ***student number*** | |
| | student1 | Jim | stu no1 | |
| | student2 | Gill | stu no2 | |
| | student3 | Mike | stu no3 | |
| | student4 | Rudy | stu no4 | |
| | student5 | Martin | stu no 5 | |
| | student6 | Shirley | stu no 6 | |
| | student7 | Tracy | stu no 7 | |
| | student8 | Keith | stu no 8 | |
| cs | *instances* | *grade* | *instances* | *grade* | *instances* | *grade* |
| | cs1 | A | cs2 | B | cs3 | B |
| | cs4 | C | cs5 | C | cs6 | B |
| | cs7 | A | cs8 | F | cs9 | B |
| | cs10 | D | cs11 | C | cs12 | B |
| | cs13 | B | cs14 | B | cs15 | F |
| | cs16 | A | cs17 | C | cs18 | D |
| | cs19 | C | cs20 | D | cs21 | A |
| | cs22 | B | cs23 | B | cs24 | A |
| home | *instances* | *number* | *street name* | | | |
| | home1 | 20-22 | Sunset Avenue | | | |
| | home2 | 6-7 | Avenue George V | | | |
| hostel | *instances* | *name* | | | | |
| | hostel1 | KR | | | | |
| | hostel2 | SH | | | | |
| | hostel3 | KEVII | | | | |
| tutor | *instances* | ***staff number*** | *name* | *preferred area* | | |
| | tutor1 | stf no 1 | J-Sun | SE, RE, FM, SW | | |
| | tutor2 | stf no2 | G-Dobbie | SE, DB, Network | | |
| | tutor3 | stf no 3 | M-Jackson | RE, SE, Grid | | |
| cst | *instances* | *feedback* | *instances* | *feedback* | *instances* | *feedback* |
| | cst1 | positive | cst2 | neutral | cst3 | neutral |
| | cst4 | neutral | cst5 | neutral | cst6 | neutral |
| | cst7 | positive | cst8 | negative | cst9 | neutral |
| | cst10 | negative | cst11 | neutral | cst12 | neutral |
| | cst13 | neutral | cst14 | neutral | cst15 | negative |
| | cst16 | positive | cst17 | neutral | cst18 | negative |
| | cst19 | neutral | cst20 | negative | cst21 | positive |
| | cst22 | neutral | cst23 | neutral | cst24 | positive |
| sport club | *instances* | ***name*** | *sport* | | | |
| | club1 | yachting club | yachting | | | |
| | club2 | boxing club | boxing | | | |
| | club3 | tennis club | tennis | | | |
| sm | *instances* | *join date* | *instances* | *join date* | | |
| | sm1 | aug-02-2002 | sm2 | may-15-2003 | | |
| | sm3 | jan-25-2003 | sm4 | oct-01-2003 | | |
| | sm5 | jul-19-2005 | sm6 | dec-31-2004 | | |