USING CORRESPONDENCE ASSERTIONS TO SPECIFY THE SEMANTICS OF VIEWS IN AN OBJECT-RELATIONAL DATA WAREHOUSE

Valéria Magalhães Pequeno, Joaquim Nunes Aparício
Depto. de Informática, Centro de Inteligência Artificial, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
Quinta da Torre 2829 516, Caparica, Portugal

Keywords: integrated information, object view, object-relational model, data warehouse.

Abstract: An information integration system provides a uniform query interface for collecting of distributed and heterogeneous, possibly autonomous, information sources, giving users the illusion that they interrogate a centralized and homogeneous information system. One approach that has been used for integrating data from multiple databases consists in creating integrated views, which allows for queries to be made against them. Here, we propose the use of Correspondence Assertions (CAs) to formally specify the relationship between the integrated view schema and the source database schemas. In this way, CAs are used to assert that the semantic of some schema’s components are related to the semantic of some components of another schema. Our formalism has the advantages of proving a better understanding of the semantic of integrated view, and of helping to automate some aspects of data integration.

1 INTRODUCTION

An Integration Information (II) system provides a uniform interface for querying collections of pre-existing data sources that were created independently. In the recent years, the number of applications requiring integrated access to several distributed, heterogeneous information sources has immensely increased. A wide range of techniques has been developed to address the problem of information integration in databases (Zhou et al., 1996; Goasdoué et al., 2000).

Basically, there are two approaches to data integration: the virtual integrated views (Batini et al., 1986) and the materialized integrated views (Zhou et al., 1996). In the first one, data exits in the local sources and the II system must reformulate the queries submitted to it, at run time, into queries against the source schemas. The results from these queries on the local sources are translated, filtered and merged to form a global result and finally, the final answer is returned to the user. Whereas, in the materialized approach, information from each source database is extracted in advance and then translated, filtered, merged and stored in a centralized repository, called Data Warehouse (DW). Thus, when the user’s query arrives, it can be evaluated directly at the repository, and no access to the source databases is required.

There are some problems in integrating information from multiple information sources. One of difficulties is that, normally, the data have heterogeneous structure and content. In this case, it is usual that the integration systems are based on the specification of a single integrated schema describing a domain of interest. Additionally, there is a set of source “descriptions” (mappings) expressing how the content of each source available to the system is related to the domain of interest.

In our work, we used a formal object-relational data model (Pequeno and Aparício, 2003) for modeling the integrated view schema and source database schemas of a data warehousing environment. We propose the use of the Correspondence Assertions (CAs) to formally specify the relationship between the view schema and the source database schemas. CAs are a special kind of integrity constraints that are used to assert the correspondence among schema components. In our research, we use the view CAs to semantically elucidate how the view objects are synthesized from the source class objects. Our formalism is advantageous in proving a better understanding of the semantics of integrated view, and in helping to automate some aspects of data integration. In this paper we focus on how the CAs can be used for helping to generate the integrated view definition.
Correspondence assertions to point out the semantics of views already were developed for other works (Lóscio, 1998; Vidal and Pequeno, 2000; Vidal et al., 2001; da Costa, 2002). We extend the work of (Vidal and Pequeno, 2000) to contemplate relational structures and aggregation functions. In (da Costa, 2002) the CAs are used to assert the correspondence among the view schema and the local sources schemas, where these local sources schemas can be relational or object-relational ones (like in our work). However, to best of our knowledge, our paper is the first one to specify CAs in Object-Relational(OR)data warehousing environment, and to consider CAs between properties with aggregations functions.

The remainder of this paper is organized as follows. The next Section presents our formal model\(^{1}\). Section 3 presents the formalism we use to assert the relationship between the integrated view schema and the source database schemas. Section 4 shows as CA can be used to define the integrated view. Section 5 is devoted to present some previous approaches and contrast them with ours. Finally, Section 6 concludes, pointing out future work.

2 TERMINOLOGY

In this section, we present the basic concepts of the object model used to represent the integrated view schema and the source database schemas. This model was based in Object-Oriented Data Model (OODM) standard ODMG3 (Cattell et al., 2000), but we tried to preserve the main characteristics of Relational Data Model (RDM) proposed by Codd in (Codd, 1970).

In accordance with the object model described in ODMG3, we distinguish objects from literals as follows: objects represent real world entities and have a unique identifier (OID), while literals are special types of objects that have no identifier.

An object is an instance of a type. Thus, types serve as templates for their instances. In our model we define some types, namely: base (integer, float, string and boolean), reference, tuple and collections (set, list and array). The tuple type is an important type because it can represent the relation schema in RDMs.

The component of the type tuple consists of properties, which can be classified into attributes and relationships. The domain of an attribute is a literal or a collection of literals. On the other hand, the domain of a relationship is an object or a collection of objects. Properties also can be classified into singlevalued and multivalued. A property is denoted singlevalued when each instance of its type can be related to at most one object (or literal) of the property domain. A property is denoted multivalued when each instance of its type can be associated to many instances of the property domain. We consider the properties whose types are base, reference or tuple as singlevalued properties and the properties whose types are collections (set, list or array) as multivalued properties.

We distinguish types from classes. A class is a set of objects that is associated with a type. We distinguish two kinds of classes: the object classes, whose instances of the type are objects; and the literal classes, whose instances of the type are literals.

It is common to present classes in diagrams\(^{2}\). In Figure 1, the classes EMPLOYEE, DIVISION, MANAGER and GOOD are represent as rectangles. The attributes with their types are inside the rectangles. Single arrows represent single valued relationships and double arrows represent multi valued relationships.

Figure 1: An object-relational schema.

An OR schema is a set of class definitions that serve as templates to generate the application domain objects. It is important to note that an OR schema can be only a relational schema or an OR schema.

All classes in an OR schema have a distinct name, a structured type, a finite set of signatures (the methods) and an extension. The latter consists of a set of objects that are members of a class at a given moment. An instance of an OR schema \(S\) populates object classes with OIDs, assigns values to the OIDs, assigns values to literal class names, and assigns semantics to the methods signatures.

Objects can be related through paths connecting two or more properties. From Figure 1, one can observe that an employee is related to his/her division manager through a path \(\text{div} \circ \text{mgr}\). We distinguish

\(^{1}\)Only basic concepts are showed. For more details the reader can refer to (Pequeno and Aparício, 2003).

\(^{2}\)The graphic notation is based on Unified Modeling Language (UML) and ODMG3.
two kinds of paths: a reference path and a value path, defined as:

Let \( C \) be a finite set of class names, \( \mathcal{P} \) be a set of properties names, \( \mathcal{T} \) be a set of all types, \( \text{props}(C) \) refers the set of properties defined for a class \( C \) and \( \text{dom}(\tau) \) be the mapping that attaches to every type \( \tau \) a corresponding value set (domain).

**Definition 1** (Reference path of a class) Let \( C_1, C_2, \ldots, C_{n+1} \in \mathcal{C}, p_1, \ldots, p_n \) be properties in \( \mathcal{P} \) and \( \tau_1, \ldots, \tau_n \) be types in \( \mathcal{T} \) such that \( p_i: \tau_i \in \text{props}(C_i), 1 \leq i \leq n, p_1 \circ p_2 \circ \cdots \circ p_n \) is a reference path of \( C_i \) iff \( \text{dom}(\tau_i) = C_{i+1}, 1 \leq i \leq n. \)

This means that instances of \( C_1 \) are related with the instances of \( C_{n+1} \) through the reference path \( p_1 \circ p_2 \circ \cdots \circ p_n. \)

**Definition 2** (Value path of a class) Let \( C_1, C_2, \ldots, C_{n+1} \in \mathcal{C}, p_1, \ldots, p_n \) be properties in \( \mathcal{P} \) and \( \tau_1, \ldots, \tau_n \) be types in \( \mathcal{T} \) such that \( p_i: \tau_i \in \text{props}(C_i), 1 \leq i \leq n, p_1 \circ p_2 \circ \cdots \circ p_n \) is a value path of \( C_i \) iff \( \text{dom}(\tau_i) = C_{i+1}, 1 \leq i \leq n - 1 \) and \( \text{dom}(\tau_n) = w, \) where \( w \) is a constant or a collection of constants (integer, float, string or boolean values).

This means that the instances of \( C_1 \) are related with the value \( w \) (with domain of \( \tau_n \)) of some property from \( C_n \) through the value path \( p_1 \circ p_2 \circ \cdots \circ p_n. \)

In Figure 1, \( \text{div}_1 \circ \text{mgr}_1 \) is a reference path (for class \( \text{EMPLOYEE} \)) and \( \text{div}_1 \circ \text{mgr}_1 \circ \text{managerName} \) is a value path (for class \( \text{EMPLOYEE} \)).

Now, we extend this model with the view classes and view schema concepts, as follows. An view class is defined as an object class derived from one or more base classes, called base classes (in a base schema). The view class objects can be physically stored in a database or not, and these objects can be the matching of relational data or object-relational data from local sources.

An integrated view schema, or simply view schema, is formed by the set of view classes (from one or more base schemas) and this view schema is independent from its underlying schemas.

## 3 Correspondence Assertions

The Correspondence Assertions (CAs) of a view class formally specify the relationships between the view class and its base classes. In this way, CAs are used to assert that the semantic of some schema components are related to the semantic of some components of another schema.

In our work, the relationships between the integrated view schema and the source database schemas can be specified by the following four kind of CAs:

### Extension Correspondence Assertion (ECA)

The ECAs are used to specify the relationship that exists between the extension view class and extensions base classes. Thus, the ECAs are used to define which objects of the base classes should have a corresponding semantically equivalent object in the view class. Two objects \( o_1 \) and \( o_2 \) are semantically equivalent \( (o_1 \equiv o_2) \) if \( o_1 \) and \( o_2 \) represent the same object in the real world.

We define the root classes of a view class \( V \) as all the classes that are related to \( V \) through some ECA. Consider, for example, the view class \( \text{STUDENT}_v \) (see Figure 2), which contain informations about all students in a university, including his/her salary if the student also works.

![Figure 2: The integrated view schema \( V_1. \)](image)

### Object Correspondence Assertion (OCA)

Object Correspondence Assertions (OCAs) are used to define which objects of the base classes should have a corresponding object in the view class. Two objects \( o_1 \) and \( o_2 \) are object equivalent \( (o_1 \equiv o_2) \) if \( o_1 \) and \( o_2 \) represent the same object in the real world.

We define the root classes of a view class \( V \) as all the classes that are related to \( V \) through some OCA. Consider, for example, the view class \( \text{STUDENT}_v \) (see Figure 2), which contain informations about all students in a university, including his/her salary if the student also works.

![Figure 3: The source database schema \( DB_2. \)](image)

### Property Correspondence Assertion (PCA)

Property Correspondence Assertions (PCAs) are used to define which properties of the base classes should have a corresponding property in the view class. Two properties \( p_1 \) and \( p_2 \) are property equivalent \( (p_1 \equiv p_2) \) if \( p_1 \) and \( p_2 \) represent the same property in the real world.

We define the root classes of a view class \( V \) as all the classes that are related to \( V \) through some PCA. Consider, for example, the view class \( \text{STUDENT}_v \) (see Figure 2), which contain informations about all students in a university, including his/her salary if the student also works.

### Path Correspondence Assertion (PCA)

Path Correspondence Assertions (PCAs) are used to define which paths of the base classes should have a corresponding path in the view class. Two paths \( p_1 \) and \( p_2 \) are path equivalent \( (p_1 \equiv p_2) \) if \( p_1 \) and \( p_2 \) represent the same path in the real world.

We define the root classes of a view class \( V \) as all the classes that are related to \( V \) through some PCA. Consider, for example, the view class \( \text{STUDENT}_v \) (see Figure 2), which contain informations about all students in a university, including his/her salary if the student also works.
ψ2: STUDENT ⊕ EMPLOYEE. ψ2 specifies that STUDENT and STUDENT have equivalent properties, i.e., for each STUDENT object there is one semantically equivalent object in STUDENT, and vice-versa. ψ2 specifies that the classes STUDENT and EMPLOYEE can have objects in common.

In accordance with the kind of ECA relating a view class with its root classes, we distinguish six different kinds of view classes: equivalence, selection, union, difference, intersection and generalization.

### 3.2 Object CAs

The OCAs specify the matching function that exists between the objects of a class with the object of another class. These assertions define the conditions in which an object of a class is semantically equivalent to an object of another class.

In the case of ECAs, given two classes C1 and C2 to any state D there is a mapping function that defines an one to one correspondence between the objects of C1 and C2. We can have others mapping functions, maybe one that makes the relationship among several objects of a class with one object of another class. This is the case, for example, of the view class with aggregation functions.

Object matching (Doan et al., 2003) is an important aspect of data integration and it can be expensive to compute. Most systems assume that a universal key is available for performing object matching. In this work, we do not address this problem and, as proposed in (Zhou et al., 1996), we assume that match criteria is defined by a high-level mechanism. In case of view classes without aggregations it defines 1:1 correspondences between the objects in families of corresponding classes.

### 3.3 Property CAs and Path CAs

The PrCAs and the PaCAs specify how the properties values of the view class objects are derived from properties values of their root classes objects. For instance, the property salary of the view class ST&EMP, (see Figure 2) is defined by the PrCA: ST&EMP, salary ≡ EMPLOYEE.salary1, which specifies that given an instance s of ST&EMP, if there is an instance e in EMPLOYEE such s ≡ e, then s.salary = e.salary1. The property manager of the class ST&EMP is defined by the PaCA ST&EMP,.manager ≡ EMPLOYEE.division•manager1, which specify that given an instance s of ST&EMP, if there is an instance e in EMPLOYEE such s ≡ e, then s.manager = e.division•manager1. Note that a view class property can be associated with more than one PrCA and/or PaCA. For instance, the property name of the view class ST&EMP has the following PrCAs:

\[
\psi: \text{ST}\&\text{EMP}_v, \text{name}_v \equiv \text{EMPLOYEE}\_\text{employeeName}_1 \land \text{ST}\&\text{EMP}_v, \text{name}_v \equiv \text{STUDENT}\_\text{name}_2.
\]

This means that the values of name can be derived from employeeName1 and name2.

In a data warehousing environment, it is common to have materialized views involving aggregation, because clients of DW often want to summarize data in order to analyze trends (Gray et al., 1995; V. Narayanan, 1996). Thus, we define some PrCAs whose properties values are gotten by aggregation functions. These PrCAs are different from other CAs, for the reason that there is no one to one correspondence between the view schemas and the source database schemas. Instead of this, there is a mapping of one view class object to many root class objects.

At this point, we must extend the definition of a root class to include the classes with aggregation. Thus, we define the root classes of a view class V as the set of all classes that are related to V through some ECA or some PrCA with aggregation function.

In our work, the aggregation functions mentioned are ones supported by the most of the queries languages, like SQL-3 (Fortier, 1999). The statistic aggregation functions, as standard deviation and variance, are not considered in our work, although they are supported in some OR databases (for instance, the reader can refer to the work of (Chamberlin, 1996)).

We can distinguish six kinds of PrCAs with aggregation: sum (summation), count (minimum), max (maximum), avg (average) and group-by. In Figure 2, we can observe the view classes: 1) ST_GOOD, which is a set of a sole object and which is related to root class good through the PrCAs with aggregation, such that \(\psi: \text{ST}\_\text{GOOD}_v, \text{quantity}_v \equiv \text{count}(\text{GOOD})\) and \(\psi: \text{ST}\_\text{GOOD}_v, \text{sumPrice}_v \equiv \text{sum}(\text{GOOD}.\text{sentPrice})\); and 2) ST_COURSE, which is a set of objects and is related to root class COURSE through the PrCA with aggregation of group-by presented in Figure 4.

\[
\psi: \text{ST}\_\text{COURSE}_v(\text{course}_v, \text{nameCourse}_v, \text{students}_v) \equiv \text{COURSE}(\text{M}[\text{name}_2], \text{A}[\text{cod}_2], \text{F}[\text{(count.std2)}]) \land \text{course}_v \rightarrow \text{COURSE}.\text{code}_2 \land \text{nameCourse}_v \rightarrow \text{COURSE}.\text{name}_2 \land \text{students}_v \rightarrow \text{count}(\text{COURSE}.\text{std}_2)
\]

Figure 4: A property CA with aggregation of group-by between ST_COURSE and COURSE.

In a PrCA with aggregation of group-by there are notations that need some explanation. Thus, we can specify this type of PrCA as following: \(V(p_1, ..., p_m, p’n_1, ..., p’n_k) \equiv (\text{C} | \mathcal{M}[p_1, p_2, ..., p_m], A[p’n_1, p’n_2, ..., p’n_k], \mathcal{F}[f_1, f’n_1], f_2, f’n_2, ...)\).
4 USING CAs TO DEFINE VIRTUAL VIEW CLASSES

In our approach, the integrated view schema of a DW consists of three steps:

1. **Integrated view modeling** - Analyzes the requirements and specifies the integrated view schema using a high-level data model. In this work, we use our OR data model to represent the integrated view schema (Pequeno and Aparício, 2003). The graphic notation used is based on UML and ODMG 3.

2. **View correspondence assertions generation** - Combines the integrated view schema with the local schemas in order to identify the CAs that formally assert the relationships between the integrated view schema and the source schemas. To achieve this, all schemas should be expressed in the same data model (called “common” data model).

3. **Integrated view definition** - Generates the integrated view definition based on the integrated view schema and the view CAs. The integrated view definition consists of a set of queries, when the view classes are virtual, and a set of rules that maintain the view classes to reflect updates occurred in root classes, when the view classes are materialized.

To illustrate our approach, consider the integrated view schema $V_1$ in Figure 2, which integrates information from employees in $DB_1$ (see Figure 1) and students in $DB_2$ (see Figure 3).

The next step in the process of building the integrated view $V_1$ is generating the view correspondence assertions. This process consists of following steps:

1. Identify the Extension CAs.
2. For each ECA $\psi$ identified in step 1, where $\psi$ relates the view class $V_1$, whose objects are of the type $\tau_{v_1}$, to root class $C_1$, whose objects are of the type $\tau_{c_1}$, do:
   (a) Identify the Object CAs from view class objects $V_1$ to objects in $C_1$.
   (b) Compare the types $\tau_{c_1}$ and $\tau_{v_1}$. In this step the types $\tau_{c_1}$ and $\tau_{v_1}$ are compared and the Properties CAs and Path CAs are identified.
3. Identify the Properties CAs with aggregation.
4. For each Property CA with aggregation $\Psi$ identified in step 3, where $\Psi$ relates the view class $V_2$ to root class $C_2$, do:
   (a) Identify the matching function between view class objects $V_2$ to objects in $C_2$.

Some examples of PrCAs are shown in Figure 5.

![Figure 5: Some view PrCAs among student, and student](image-url)
create row type TSTUDENT,v
(
  number_v, int,
  name_v, char(30),
  identify_v, char(11),
  telephone_v, set(char(9)),
  birthday_v, char(8),
  salary_v, float,
);

Figure 6: STUDENT_v type of view class definition.

create view STUDENT_v of TSTUDENT_v
as select S.number_2, S.name_2, S.identity_2,
       S.telephone_2, S.birthday_2, E.salary_1
from STUDENT S in DB_2 left outer join
     EMPLOYEE E in DB_1
on S.identity_2 = E.identity_1

Figure 7: STUDENT_v view class definition.

Figure 7 presents the view class
STUDENT_v. It extracts information about employees
and students from the local sources DB_1 and DB_2,
respectively. Keep attention to the fact that the view
class STUDENT_v is of the type TSTUDENT_v (see Figure
6). As we can observe, the clause “from” of the def-
inition for STUDENT_v correctly implements an equiva-
lence view as specified by the ECA ψ_1, which has
some objects in common with another class (as speci-
fied by the ECA ψ_2). In this case, the equivalence view
represents a left outer join view. The clause “select”
of this definition is denoted based on PrCAs such that:
ψ_9, ψ_10 and ψ_11 (see Figure 5). The clause “on” of
this definition is denoted based on an object CA be-
tween STUDENT and EMPLOYEE.

5 RELATED WORK

There is large extension of related literature on in-
formation integration in databases, such that: Her-
mes(Subrahmanian et al., 1995) and Garlic(Carey
et al., 1995). The focus of all these systems are on
building a data integration architecture based on me-
diators(Wiederhold, 1992). The mediator concept is
slightly different from the DW. Mediators, normally,
are built to provide an integrated and transparent ac-
cess to heterogeneous and possibly distributed data
sources, and primarily are used in operational data en-
vironment. DWs provide an integrated access to data
derived from operational data and primarily are used
to support the decision-making activities.

The majority of the works found in the litera-
ture focus on relational DWs(Iqbal et al., 2003)).
To the best of our knowledge, there is only one work closely related to ours: the Object-Relational
Data Warehousing System (ORDAWA)(Czejdo et al.,
2001). Their approach, with regard to the problem
of integrating of heterogeneous data in a DW con-
sists on: 1) definition of an OO view schema; 2) de-
velopment of data structures called: Class Mapping
Structure(CMS), Object Mapping Structure(OMS),
and Log. CMS is used to store derivation links be-
tween the view classes and their root classes. OMS is
used to identify the object matching between the view
classes and their root classes. Log is used to record
modifications made to root classes objects.

The aim of the data structure OMS in ORDAWA
is the same of Object CAs: to indicate when two ob-
jects are the same in the real world. Already the role
of CMS in ORDAWA is like our ECAs, PrCAs and
PaCAs, but our CAs are better in following aspects:
• They give us a clear notions of the relationship be-
tween the integrated view schema and the source
database schemas, i.e., the designer/user has a bet-
ter understanding of the semantics associated with
the integrated view;
• They can assist the process of generating the in-
tegrated view definition;
• They give us a high-level and language-indepen-
dent specification of an integrated view.

Seemingly, both approaches mention the same
kinds of view classes, but an issue addressed in this
paper that has not been addressed in ORDAWA is
the support to view classes with aggregation function,
which is an important issue in DW environment.

6 CONCLUSIONS

In this paper, we propose the use of Correspondence
Asser tions (CAs) to formally assert the relationship
between the integrated view schema and the source
database schemas. An advantage of using CAs is that
they allow for the specification of the integrated view
in a formal and language-independent way. More-
over, they provide designers/users a better under-
standing of the semantic associated with the inte-
grated view.

We have showed how the CAs can be used for
aiding the generation of the integrated view defini-
tion. This process was illustrated with some exam-
pies showing how to generate queries (when the
view classes are virtual) based on the integrated view
schema and view CAs.
As future work, we will investigate how the CAs can be used to automate the maintenance of the integrated view, when the view classes are materialized. Another important direction for future work is the development of an object-relational algebra to specify view classes. Additionally, we intend to extend our data model to contemplate view schema and view class definitions.

REFERENCES


