# 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. Additionaly, 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.



Magalhães Pequeno V. and Nunes Aparício J. (2005).
USING CORRESPONDENCE ASSERTIONS TO SPECIFY THE SEMANTICS OF VIEWS IN AN OBJECT-RELATIONAL DATA WAREHOUSE.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 219-225
DOI: 10.5220/0002507602190225

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, namelly: *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.

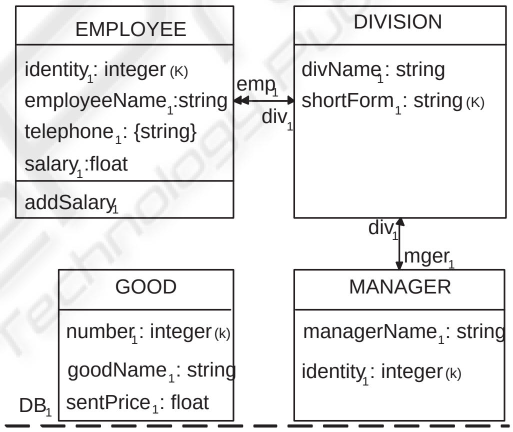


Figure 1: An object-relational schema.

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 $\mathbf{div}_1 \bullet \mathbf{mger}_1$. 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 $\mathcal{C}$ be a finite set of class names, $\mathcal{P}$ be a set of properties names, $\mathcal{T}$ be a set of all types, *props*(**C**) refers the set of properties defined for a class **C** and ***dom***($\tau$) be the mapping that attaches to every type a corresponding value set (domain).

**Definition 1 (***Reference path of a class)* *Let* $\mathbf{C}_1$, $\mathbf{C}_2$, *...,* $\mathbf{C}_{n+1}$ *in* $\mathcal{C}$, $\boldsymbol{p}_1$, *...,* $\boldsymbol{p}_n$ *be properties in* $\mathcal{P}$ *and* $\tau_1$, *...,* $\tau_n$ *be types in* $\mathcal{T}$ *such that* $\boldsymbol{p}_i$:$\tau_i \in$ ***props***($\mathbf{C}_i$), $1 \le i \le n$. $\boldsymbol{p}_1 \bullet \boldsymbol{p}_2 \bullet ... \bullet \boldsymbol{p}_n$ *is a reference path of* $\mathbf{C}_1$ *iff* ***dom***($\tau_i$)= $\mathbf{C}_{i+1}$, $1 \le i \le n$. □

This means that instances of $\mathbf{C}_1$ are related with the instances of $\mathbf{C}_{n+1}$ through the reference path $\boldsymbol{p}_1 \bullet \boldsymbol{p}_2 \bullet ... \bullet \boldsymbol{p}_n$.

**Definition 2 (***Value path of a class)* *Let* $\mathbf{C}_1$, $\mathbf{C}_2$, *...,* $\mathbf{C}_{n+1}$ *in* $\mathcal{C}$, $\boldsymbol{p}_1$, *...,* $\boldsymbol{p}_n$ *be properties in* $\mathcal{P}$ *and* $\tau_1$, *...,* $\tau_n$ *be types in* $\mathcal{T}$ *such that* $\boldsymbol{p}_i$:$\tau_i \in$ ***props***($\mathbf{C}_i$), $1 \le i \le n$. $\boldsymbol{p}_1 \bullet \boldsymbol{p}_2 \bullet ... \bullet \boldsymbol{p}_n$ *is a value path of* $\mathbf{C}_1$ *iff* ***dom***($\tau_i$)= $\mathbf{C}_{i+1}$, $1 \le i \le n-1$ *and* ***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 $\mathbf{C}_1$ are related with the value ***w*** (with domain of $\tau_n$) of some property from $\mathbf{C}_n$ through the value path $\boldsymbol{p}_1 \bullet \boldsymbol{p}_2 \bullet ... \bullet \boldsymbol{p}_n$.

In Figure 1, $\mathbf{div}_1 \bullet \mathbf{mger}_1$ is a reference path (for class EMPLOYEE) and $\mathbf{div}_1 \bullet \mathbf{mger}_1 \bullet \mathbf{managerName}_1$ is a value path (for class 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 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 relationship between the integrated view schema and the source database schemas can be specified by the following four kind of CAs: Extension Correspondence Assertion (ECA), Object Correspondence Assertion (OCA), Property Correspondence Assertion (PrCA) and Path Correspondence Assertion (PaCA), described below. A formal definition of these CAs can be found in (Pequeno and Aparício, 2004).

## 3.1 Extension CAs

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 $\mathbf{o}_1$ and $\mathbf{o}_2$ are semantically equivalent ($\mathbf{o}_1 \equiv \mathbf{o}_2$) if $\mathbf{o}_1$ and $\mathbf{o}_2$ 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 $\mathrm{STUDENT}_v$ (see Figure 2), which contain informations about all students in a university, including his/her salary if the student also works.

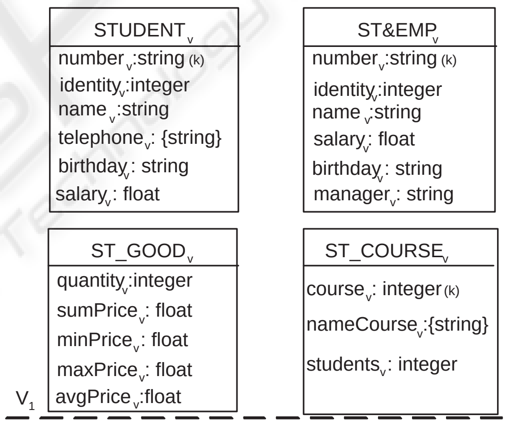


Figure 2: The integrated view schema $\mathbf{V}_1$.

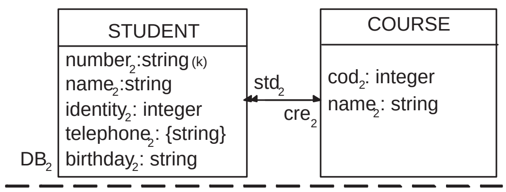


Figure 3: The source database schema $\mathbf{DB}_2$.

The $\mathrm{STUDENT}_v$ root classes are STUDENT in $\mathbf{DB}_2$ (see Figure 3) and EMPLOYEE in $\mathbf{DB}_1$ (see Figure 1), which are related to $\mathrm{STUDENT}_v$ through the following ECAs: $\psi_1$: $\mathrm{STUDENT}_v \equiv \mathrm{STUDENT}$ and

$\psi_2$: STUDENT$_v \oslash$ EMPLOYEE. $\psi_1$ specifies that STUDENT$_v$ and STUDENT are equivalent, i.e., for each STUDENT object there is one semantically equivalent object in STUDENT$_v$, and vice-versa. $\psi_2$ specifies that the classes STUDENT$_v$ 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 $\mathbf{C}_1$ and $\mathbf{C}_2$ to any state $\mathcal{D}$ there is a mapping function that defines an one to one correspondence between the objects of $\mathbf{C}_1$ and $\mathbf{C}_2$. 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 assumes 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**$_v$ of the view class ST&EMP$_v$ (see Figure 2) is defined by the PrCA: ST&EMP$_v$.**salary**$_v \equiv$ EMPLOYEE.**salary**$_1$, which specifies that given an instance **s** of ST&EMP$_v$, if there is an instance **e** in EMPLOYEE such **s** $\equiv$ **e**, then **s**.**salary**$_v$ = **e**.**salary**$_1$. The property **manager**$_v$ of the class ST&EMP$_v$ is defined by the PaCA ST&EMP$_v$.**manager**$_v \equiv$ EMPLOYEE.**div**$_1 \bullet$ **mger**$_1 \bullet$ **managerName**$_1$, which specify that given an instance **s** of ST&EMP$_v$, if there is an instance **e** in EMPLOYEE such **s** $\equiv$ **e**, then **s**.**manager**$_v$ = **e**.**div**$_1$.**mger**$_1$.**managerName**$_1$. Note that a view class property can be associated with more than one PrCA and/or PaCA. For instance, the property **name**$_v$ of the view class ST&EMP$_v$ has the following PrCAs: ST&EMP$_v$.**name**$_v \equiv$ EMPLOYEE.**employeeName**$_1$ and ST&EMP$_v$.**name**$_v \equiv$ STUDENT.**name**$_2$. This means that the values of **name**$_v$ can be derived from **employeeName**$_1$ and **name**$_2$.

In a data warehousing environment, it is common to have materialized views involving aggregation, because clients of DWs often want to summarize data in order to analyze trends (Gray et al., 1995; V. Harinarayan, 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***, ***min*** (minimum), ***max*** (maximum), ***avg*** (average) and ***group-by***. In Figure 2, we can observe the view classes: I) ST_GOOD$_v$, which is a set of a sole object and which is related to root class GOOD through the PrCAs with aggregation, such that $\Psi_6$:ST_GOOD$_v$.**quantity**$_v \cong$***count***(GOOD) and $\Psi_7$:ST_GOOD$_v$.**sumPrice**$_v \cong$***sum***(GOOD.**sentPrice**$_1$); and II) ST_COURSE$_v$, 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.

$$\begin{aligned}\Psi_8{:}\text{ST\_COURSE}_v(&\mathbf{course}_v,\ \mathbf{nameCourse}_v,\ \mathbf{students}_v) \cong \\ &\text{COURSE}(\mathcal{M}[\mathbf{name}_2]\mathcal{A}[\mathbf{cod}_2]\mathcal{F}[(\boldsymbol{count},\mathbf{std}_2)] \wedge \\ &\mathbf{course}_v \rightarrow \text{COURSE}.\mathbf{cod}_2 \wedge \\ &\mathbf{nameCourse}_v \rightarrow \text{COURSE}.\mathbf{name}_2 \wedge \\ &\mathbf{students}_v \rightarrow \boldsymbol{count}(\text{COURSE}.\mathbf{std}_2)\end{aligned}$$

Figure 4: A property CA with aggregation of group-by between ST_COURSE$_v$ 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: $\mathbf{V}(\boldsymbol{p}_1, ..., \boldsymbol{p}_n, \boldsymbol{p'}_1, ..., \boldsymbol{p'}_m, \mathbf{p''}_1, ..., \mathbf{p''}_k) \cong \mathbf{C}(\mathcal{M}[\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n]\mathcal{A}[\boldsymbol{p'}_1, \boldsymbol{p'}_2, ..., \boldsymbol{p'}_m]\ \mathcal{F}[(f_1, \mathbf{p''}_1), (f_2, \mathbf{p''}_2), ...,$

($f_k$, **p**"$_k$)]) $\wedge$ **V**.$\boldsymbol{p}_1 \rightarrow$ **C**.$\boldsymbol{p}_1 \wedge ... \wedge$ **V**.$\boldsymbol{p}_n \rightarrow$ **C**.$\boldsymbol{p}_n$ $\wedge$ **V**.***p'***$_1 \rightarrow$ **C**.***p'***$_1 \wedge ... \wedge$ **V**.***p'***$_m \rightarrow$ **C**.***p'***$_m \wedge$ **V**.**p**"$_1$ $\rightarrow f_1$(**C**.**p**"$_1$) $\wedge$ ... **V**.**p**"$_k \rightarrow f_k$(**C**.**p**"$_k$), where:

1. $\mathcal{M}$[$\boldsymbol{p}_1$, $\boldsymbol{p}_2$,..., $\boldsymbol{p}_n$], with $\boldsymbol{p}_i$:$\tau_i \in$ *props*(**C**), $1 \leq i \leq n$, is a list of properties of the root class defined in **C**. Pay attention to $\mathcal{M}$ can be an empty list.
2. $\mathcal{A}$[***p'***$_1$, ***p'***$_2$, ..., ***p'***$_m$], with ***p'***$_i$:$\tau'_i \in$ *props*(**C**), $1 \leq i \leq m$, is a list of grouping properties of the root class specified in **C**.
3. $\mathcal{F}$[($f_1$, **p**"$_1$), ($f_2$, **p**"$_2$), ..., ($f_k$, **p**"$_k$)], with **p**"$_i$:$\tau$"$_i$ $\in$ *props*(**C**), $1 \leq i \leq k$, is a list of ($\langle$*function*$\rangle$, $\langle$*property*$\rangle$) pairs. In each pair, $\langle$*function*$\rangle$ is one of the allowed functions - such as ***sum, avg and max*** - and $\langle$*property*$\rangle$ is a property of the root class defined by **C**. Observe that $\mathcal{F}$ can be an empty list.

# 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 ODMG3.
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 $\mathbf{V}_1$ in Figure 2, which integrates information from employees in $\mathbf{DB}_1$ (see Figure 1) and students in $\mathbf{DB}_2$ (see Figure 3).

The next step in the process of building the integrated view $\mathbf{V}_1$ 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 $\mathbf{V}_1$, whose objects are of the type $\tau_{v1}$, to root class $\mathbf{C}_1$, whose objects are of the type $\tau_{c1}$, do:
   (a) Identify the Object CAs from view class objects $\mathbf{V}_1$ to objects in $\mathbf{C}_1$.
   (b) Compare the types $\tau_{c1}$ and $\tau_{v1}$. In this step the types $\tau_{c1}$ and $\tau_{v1}$ 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 $\mathbf{V}_2$ to root class $\mathbf{C}_2$, do:
   (a) Identify the matching function between view class objects $\mathbf{V}_2$ to objects in $\mathbf{C}_2$.

   Some examples of PrCAs are shown in Figure 5.

$\Psi_9$:STUDENT$_v$.**number**$_v$ $\equiv$STUDENT.**number**$_2$
$\Psi_{10}$:STUDENT$_v$.**identity**$_v$ $\equiv$STUDENT.**identity**$_2$
$\Psi_{11}$:STUDENT$_v$.**name**$_v$ $\equiv$STUDENT.**name**$_2$

Figure 5: Some view PrCAs among STUDENT$_v$ and STUDENT

The final step in the process of building an integrated view is the generation of the integrated view definition. In our approach, the integrated view is defined based on the integrated view schema and the view correspondence assertions.

As we already mentioned, a view class **V** can be implemented as a virtual class as well as a materialized one. If **V** is virtual, then the definition of this class is determined from its root classes using its CAs. Otherwise, it is materialized and an extension to this class is explicitly created and rules to maintain **V** (to reflect updates occurred in its root classes) are generated. In this paper, we only present definitions to the queries to virtual view classes. Mechanisms to maintain materialized view classes are going to be investigated in the next stage of our research.

During the search for a query representation, several queries languages (SQL-3(Fortier, 1999), ODMG OQL(Cattell et al., 2000) and $O_2$View(dos Santos, 1994)) were considered, but none query representation has the expressivity necessary to define our view classes. Thus, we decide to use a combination of the SQL-3 and ODMG OQL query languages for the definition of our virtual view classes. SQL-3 is an OR database language, which is a model flexible enough to represent and store any data type supported by relational and OO models, as well as some in between. ODMG OQL is an OO database language and it is very close to SQL-2. Despite of these choices our approach can be used to denote virtual view classes definitions in any other languages.

*create row type* T$\text{STUDENT}_v$
(
**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: $\text{STUDENT}_v$ type of view class definition.

*create view* $\text{STUDENT}_v$ *of* T$\text{STUDENT}_v$
*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: $\text{STUDENT}_v$ view class definition.

Figure 7 presents the definition for the view class $\text{STUDENT}_v$. 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 $\text{STUDENT}_v$ is of the type T$\text{STUDENT}_v$ (see Figure 6). As we can observe, the clause "*from*" of the definition for $\text{STUDENT}_v$ correctly implements an equivalence view as specified by the ECA $\psi_1$, which has some objects in common with another class (as specified by the ECA $\psi_2$. 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: $\psi_9$, $\psi_{10}$ and $\psi_{11}$ (see Figure 5). The clause "*on*" of this definition is denoted based on an object CA between STUDENT and EMPLOYEE.

## 5 RELATED WORK

There is large extension of related literature on information integration in databases, such that: Hermes(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 mediators(Wiederhold, 1992). The mediator concept is slightly different from the DW. Mediators, normally, are built to provide an integrated and transparent access to heterogeneous and possibly distributed data sources, and primarily are used in operational data environment. DWs provide an integrated access to data derived from operational data and primarily are used to support the decision-marking activities.

The majority of the works found in the literature 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 consists on: 1) definition of an OO view schema; 2) development of data structures called: Class Mapping Structure(CMS), Object Mapping Structure(OMS), and Log. CMS is used to store derivation links between 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 objects 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 between the integrated view schema and the source database schemas, i.e., the designer/user has a better understanding of the semantics associated with the integrated view;
- They can assist the process of generating the integrated view definition;
- They give us a high-level and language-independent 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 Assertions (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. Moreover, they provide designers/users a better understanding of the semantic associated with the integrated view.

We have showed how the CAs can be used for aiding the generation of the integrated view definition. This process was illustrated with some examples 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

Batini, C. et al. (1986). A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364.

Carey, M. J. et al. (1995). Towards heterogeneous multimedia information systems: The Garlic approach. In *International Workshop on Research Issues in Data Engineering - Distributed Object Management*, pages 124–131.

Cattell, R. et al. (2000). *The object database standard ODMG 3.0*. Morgan Kaufmann Publishers.

Chamberlin, D. D. (1996). *Using the new DB2 - IBM's object-relational database system*. Morgan Kaufmann.

Codd, E. F. (1970). A relational model of data for large shared data banks. In *Communications of the ACM*, pages 377–387.

Czejdo, B. D. et al. (2001). Design of a data warehouse over object-oriented and dynamically evolving data sources. In *12th International Workshop on Database and Expert Systems Applications*, pages 128–132.

da Costa, J. P. (2002). Gerando tradutores para a atualização de banco de dados através de visões de objetos. Master's thesis, Federal University of Ceara, Brazil.

Doan, A. et al. (2003). Object matching for information integration: a profiler-based approach. In *Proc. of Workshop on Information Integration on the Web*.

dos Santos, C. (1994). Design and implementation of an object-oriented view mechanism. Technical report, Institut National de Recherche en Informatique et en Automatique, France.

Fortier, P. (1999). *SQL3 - Implementing the SQL foundation standar*. McGraw-Hill, EUA.

Goasdoué, F. et al. (2000). The use of CARIN language and algorithms for information integration: the PICSEL system. *International Journal of Cooperative Information Systems*, 9(4):383–401.

Gray, J. et al. (1995). Data cube: A relational aggregation operator generalizing group-by, cross-tab, and subtotals. Technical Report msr-tr-95-22, Microsoft.

Iqbal, S. et al. (2003). Distributed heterogeneous relational data warehouse in a grid environment. The Computing Research Repository (CoRR) cs.DC/0306109.

Lóscio, B. F. (1998). Atualização de múltiplas bases de dados através de mediadores. Master's thesis, Federal University of Ceara, Brazil.

Pequeno, V. M. and Aparício, J. N. (2003). A formal model for object-relational databases. In *Proceedings of the 5th International Conference on Enterprise Information Systems(ICEIS)*, volume 1, pages 327–333.

Pequeno, V. M. and Aparício, J. N. (2004). Using correspondence assertion to specify the semantics of views in an object-relational data warehouse. Technical report, New University of Lisbon.

Subrahmanian, V. et al. (1995). HERMES: A heterogeneous reasoning and mediator system. Technical report, University of Maryland.

V. Harinarayan, A. Rajaraman, J. U. (1996). Implementing data cubes efficiently. In *Proceedings of the ACM SIGMOD Conference*.

Vidal, V. et al. (2001). Using correspondence assertions for specifying the semantics of XML-based mediators. In *Workshop on Information Integration on the Web*, volume 3(11).

Vidal, V. and Pequeno, V. (2000). Self-maintanance of match classes in integrated views. In *Anais do XV Simpósio Brasileiro de Banco de Dados*, Brasil.

Wiederhold, G. (1992). Mediators in the architecture of future information systems. In *IEEE Computer*, volume 25(3), pages 38–49.

Zhou, G. et al. (1996). Generating data integration mediators that use materialization. *Journal of Intelligent Information Systems*, 6(2/3):199–221.