Contextualization of OODB Schemas in CROME

Olivier Caron, Bernard Carré, and Laurent Debrauwer

{Olivier.Caron, Bernard.Carre, Laurent.Debrauwer}@lifl.fr
Laboratoire d'informatique Fondamentale de Lille, UPRESA CNRS 8022
Université des Sciences et Technologies de Lille, 59655 Villeneuve d'Ascq cedex

Abstract. View mechanisms, widely used in the relational databases, pose new questions in the object model which captures much more semantics. In this paper, we will focus on inheritance and inter-objects relationships, two main semantic aspects of the object model. Like in the relational model, most of the current works about object-oriented views assume a fine granularity of the views. View classes are defined by the application of a query operator to one or two classes of the base schema. View schemas are defined as sets of view classes. These sets are explicitly chosen by the database administrator.
We present the solutions produced by the application of our CROME model. In CROME, view classes extend the descriptions of the domain objects supplied by the base schema. The relationships introduced in the base schema are shared and preserved in view schemas. By adapting them locally, each view schema contextualize these relationships. We will show that this contextualization of the base schema gives it generic properties which enforce a stronger coherence of the view schemas.

1 Introduction

The external schema level introduced in the ANSI/X3/SPARC[1] architecture, which describes the information relevant to a group of users of the database, is now integrated in all relational database systems. View mechanisms are usually used to implement such external schemas. The base or conceptual schema is, in the relational model; composed of a set of relations. Views are also relations computed by operators of the relational algebra.

Object-oriented databases (OODBs) view mechanisms adhere to the same principles as relational views ([2], [3], [4]). In the object model, views acquire the status of class. Their extension is computed from a query and they are updatable [4]. They provide a specialized interface to base classes. However, the semantics of the object model is richer than the relational one and this pose new questions. One of the most complex questions concerns the insertion of a view class in the generalization/specialization hierarchy. This insertion is generally realized by a classifier [5]. Some work about object-oriented views have introduced the notion of external schemas also called view schemas [3]. A view schema is a set of classes and view classes explicitly chosen by the administrator of the database. The classifier computes the class hierarchy which structures these view schemas.

However, these view mechanisms remain based on a fine granularity of the view classes. The class hierarchy of the base schema is not taken into account for defining view schemas. Each class of the base schema used for defining view classes is taken into account individually like in the relational model. This fine granularity of the view classes and this lack of sharing of the class hierarchy leads to view schemas that do not consider the base schema as a whole schema with its structuring. The problem of sharing the inter-objects relationships of the base schema is not handled. Consequently, view schemas do not contextualize the base schema since they contextualize neither its inter-objects relationships nor its inheritance relationships. These relationships are not used for computing the extensions of the view classes.

We present the solutions produced by the application of our CROME model ([6], [7], [8], [9]) to build OODB view schemas. In CROME, the description of the objects is extended within the view schemas [10]. The base schema introduces the domain objects, give them their common description while the view schemas extend this description of the objects for their clients needs. View schemas also use participation constraints of instances to compute the extension of their view classes. The relationships introduced by the base schema are shared within the view schemas which contextualize them by adapting them locally ([11], [12]). These contextualizations concern the inheritance and inter-objects relationships. Therefore, they give to the base schema generic properties against the view schemas.

In this paper, we focus on Enterprise Information Systems (EIS). Within an enterprise, the clients associated with the different view schemas of the EIS match generally the functions of the enterprise. Each function supplies and handles the information which is specific to it within the EIS, relatively to the tasks it must realize. This specific information is described by the view schema which corresponds to the function. The base schema describes the information common to the whole set of functions. In this paper, the different notions of the CROME model are illustrated with the example of the EIS of a products distribution company. The management function and the sales function are structuring the organization of the company. The domain objects are partners, suppliers, customers, and products which are either professional products or consumer products.

The paper is organized as follows. Section 2 introduces the base schema while section 3 presents view schemas. In section 4, we examine how the inheritance relationships are contextualized in the view schemas while in section 5, we study the contextualization of inter-objects relationships. We will show that this contextualization leads to new participation constraints. Section 6 compares our approach with related work and section 7 concludes.

2 Base schema

Within the base schema, the domain objects are introduced by a set of classes structured by the inheritance hierarchy. These classes are called domain classes and the intension of these classes owns the description of objects which is common and relevant to the whole set of functions. These common descriptions are shared by the different functions through the domain objects. The content of the base schema is not constrained in the CROME model. The base schema can own a shared representation of the domain which can be more or less rich, depending on the sharing requirements. At the very least, this schema introduces the domain classes without giving them any common descriptions.

Example: The base schema of the example is shown in Figure 1 using the graphical notation of UML [13]. The domain objects are introduced as well as their common description by a set of classes structured by the inheritance hierarchy. An inter-objects relationship between "Customer" and "Product" is also introduced.

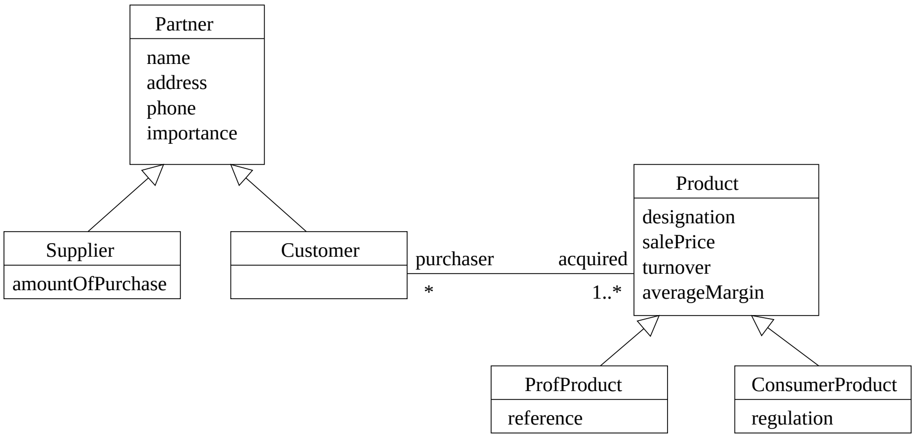

Fig. 1. Base schema

2.1 Formalization: Preliminary definitions

We define C as the set of classes of a schema

The intension of a class is the set of all its attributes. It is denoted as follows:
$\forall$ C $\in C$, intension(C) denotes the set of the attributes of the class C
The extension of a class is the set of all its instances (including those of its subclasses). It is denoted as follows
$\forall$ C $\in C$, extension(C) denotes the set of the instances of the class C
The inheritance relationship is defined as follows:

$$\forall\ (\mathrm{C}, \mathrm{C}') \in C^2,\ \mathrm{C}\ \text{subclass of}\ \mathrm{C}' \Leftrightarrow \text{intension}(\mathrm{C}) \supseteq \text{intension}(\mathrm{C}')\ \text{and}\ \text{extension}(\mathrm{C}) \subseteq \text{extension}(\mathrm{C}')$$

<u>Base-schema</u>

The base schema is composed of the set of domain classes ordered by the inheritance relationship. The set of the classes of the base schema is denoted by C_{base}.
We call O the set of all instances of the domain classes, that is the union of the extension of every class of C_{base}.

3 View schemas

We introduce now, for each function of the enterprise, a view schema. The goal of such a view schema is to describe the information relevant to its corresponding function, i.e. the information common to the whole set of functions and the information specific to this function.

3.1 Increments of description

Figure 2 shows the increments of description used to define the view schema associated with the sales function. These will be studied in the forthcoming paragraphs.

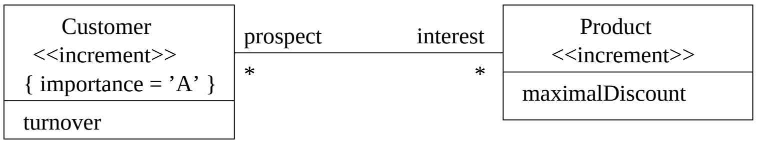

Fig. 2. Increments of description of the view schema corresponding to the sales function

From an intensional point of view, each domain object is enriched within each view schema. Its intension is extended with an increment of description relative to the specific information handled by the function corresponding to the view schema. The intension of an object is there defined as the union of its description in the base schema and this functional increment of description. Within a view schema, it is important to note that this functional increment is unique for every domain object[1].

Example: Figure 2 shows the functional increments corresponding to the sales function. We use a new UML stereotype increment in order to distinguish increments from classes. On this Figure, we can see that the intension of "Customer" object is enriched with the "turnover" attribute in the sales function. The "prospect/interest" relationship also enrichs "Customer" and "Product" objects.

We focus now on the extensional perspective of view schemas. For each function, we would like to determine the set of instances of the domain classes which

[1] By default, this increment is empty

own the information relevant to this function, and consequently participate to corresponding view schema. In order to get this set for each domain class, we propose to use a participation constraint which will select its instances. This participation constraint takes the form of a predicate, called *participation predicate*. This predicate is based exclusively on the attributes and/or relationships of the involved class introduced in the base schema. When this predicate is verified for an instance, this instance participates to the corresponding view schema, is described by the corresponding increment of description, and can own information relevant to this function. For each class of the base schema and for each view schema, there is only a participation predicate to select the instances of this class in this view schema[2].

Example: Figure 2 shows the participation constraint associated with the "Customer" object in the sales function[3]. It is a predicate based on the "importance" attribute. This attribute is introduced in the base schema as shown by Figure 1. This participation constraint means that only customers with an importance of type A are relevant to the sales function.

3.2 Status of view schemas

Each view schema extends the base schema by increasing the description of the domain objects and introducing participation constraints. So, each domain object which participates to the function corresponding to the view schema is there described by a class. The intension of this class is the result of the union of the intension of the class of the base schema and of the functional increment of description. The extension of this class is the result of the selection of instances of the class of the base schema by the participation predicate.

Such a class of a view schema is called a *view class*. The class of the base schema from which it is derived is called its *base class*. In a view schema, for each class of the base schema, there is only one corresponding view class.

Example: Figure 3 shows the "Customer" and "Product" view classes in the view schema corresponding to the sales function. These classes match the description of a customer and a product under the point of view of the sales function. The other view classes have not been shown.

Therefore, view schemas are sets of view classes. We have now to study how these sets can be organized in respect to the inheritance and inter-objects relationships, as does the base schema. We will see that, in CROME, the base schema is shared as a whole graph of classes. The relationships between classes introduced in the base schema are shared in the different view schemas which contextualize them by adapting them locally. Inheritance and inter-objects relationships are both contextualized. We will study these contextualizations in the next section.

[2] By default, this predicate is a tautology.

[3] We use the standard UML notation for constraints to denote the participation constraint.

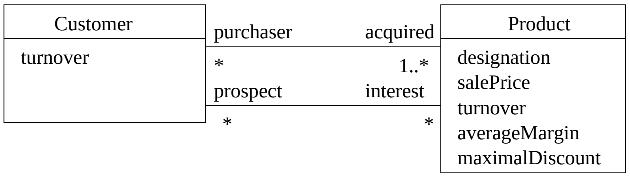

Fig. 3. View classes "Customer" and "Product" in the sales function

4 Contextualization of the inheritance relationships of the base schema

We introduce now the increments of description with their participation constraints for the management function. These are shown on Figure 4.

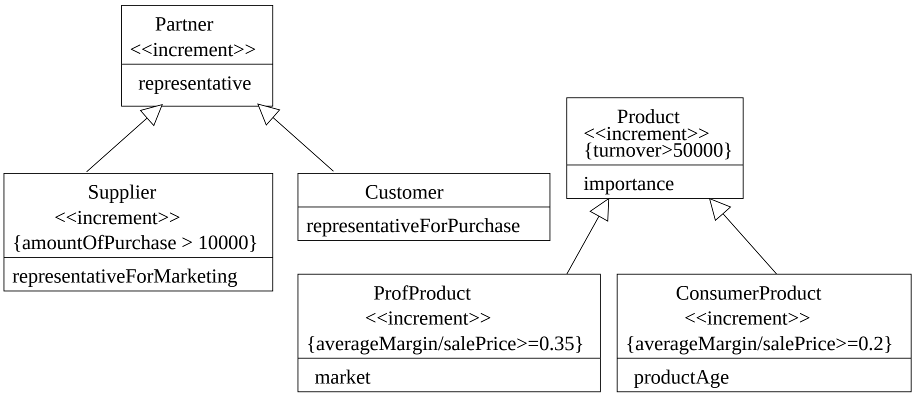

Fig. 4. Increments of description corresponding to the management function

4.1 Local inheritance of the intension of view classes

In CROME, the increment of description of a class of the base schema is inherited into the functional of description of its subclasses. Consequently, in a view schema, the intension of a view class corresponding to a class of the base schema is inherited into the intension of every view class that has for base class one of the subclasses of this class. This inheritance is realized locally in the view schema. From an intensional point of view, the inheritance relationships between classes of the base schema are contextualized in the view schemas.

Example: the increments of description of "Supplier" and "Customer" inherit from the increment of description of "Partner" in the management function as shown on Figure 4. This means that the view classes "Supplier" and "Customer" in the corresponding view schema inherit the "representative" attribute.

Formalization

Set of functions

We use the functions to distinguish each view schema among the others. F denotes the set of symbols denoting the different functions.

Intension of view classes

$\mathrm{VC_f(C)}$ denotes the view class corresponding to the class C in the function f, where $\mathrm{C} \in C_{base}$.

$\mathrm{I_f(C)}$ denotes the increment of description corresponding to the class C in the function f, where $\mathrm{C} \in C_{base}$.

intension($\mathrm{VC_f(C)}$), where $\mathrm{C} \in C_{base}$, is defined as follows:

intension($\mathrm{VC_f(C)}$) =intension(C) $\cup$ $\mathrm{I_f(C)}$

Property 1: if C is a subclass of C' and f is a function, then

intension($\mathrm{VC_f(C)}$) $\supseteq$ intension($\mathrm{VC_f(C')}$)

We have the following property

$\forall$ (C, C') $\in C^2$, C subclass of C' $\Rightarrow$ intension(C) $\supseteq$ intension(C')

Due to the inheritance of increments of description, we have also the following property:

$\forall$ (C, C') $\in C^2$, C subclass of C' $\Rightarrow$ $\mathrm{I_f(C)} \supseteq \mathrm{I_f(C')}$

Consequently, due to the definition of the intension of a view class, it is easy to prove that:

$\forall$ (C, C') $\in C^2$, C subclass of C' $\Rightarrow$ intension($\mathrm{VC_f(C)}$) $\supseteq$ intension($\mathrm{VC_f(C')}$)

4.2 Relationships between extensions of view classes

In order to verify the preservation of the inheritance relationships of the base schema within view schemas, the inheritance of the increments of description is not sufficient. We also need that the inclusion relationships between the extensions are respected within view schemas. In the base schema, a direct instance of a class belongs to the extension of all the superclasses of its class. If an instance belongs to the extension of a view class corresponding its class, it must then belong, in the same view schema, to the extension of all the view classes corresponding to the superclasses of its class.

The solution consists in giving to each instance a unique participation constraint to each view schema. This unique participation constraint is described as follows: to participate to a view schema, an instance must verify the participation predicate of the view class corresponding to its class. We have seen that, in a view schema, for each class of the base schema, there is only one corresponding view class. Since each instance has only one class in the base schema, its participation constraint is unique for each view schema. Thus, an instance that participates to a view schema belongs to the extension of the view class corresponding to its class and it belongs to the extension of the view classes corresponding to the superclasses of its class, in the manner of what occurs in the base schema.

This guarantees that the inclusion relationships between extensions of view classes are preserved in view schemas. Consequently, the inheritance relationships of the base schema are preserved in the view schemas.

Finally, this solution guarantees that, if an instance participates to a view schema, it belongs to the extension of the view class corresponding to its class. Thus, the relationship between an instance and its class is preserved in view schemas where this instance participates.

Example: in order to participate to the management function, a supplier must have an amount of purchase greater than 10000 euros (see Figure 4). This constraint is applied to the instances of "Supplier" to determine their participation to the management function, i.e. to determine if they belong to the extensions of the "Supplier" and "Partner" extensions. Figure 5 illustrates this example by showing the extension of these two classes.

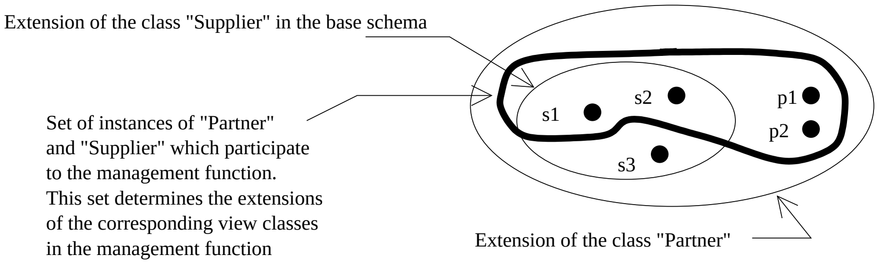

Fig. 5. Extensions of the classes "Partner" and "Supplier" and of their corresponding view classes in the management function

Within a view schema, inheritance relationships apply for both intension and extension of view classes, which are different from the intension and extension of their base classes. Each view schema preserve the inheritance relationships by adapting them locally, i.e. each view schema contextualize the inheritance relationships. This gives generic properties to the base schema. View schemas are a real adaptation of the base schema considered as a graph of classes.

Formalization

<u>Functional state of an element of O</u>

$\forall$ o $\in$ O, class(o) denotes the class of o. This class belongs to C_{base}.
Pred$_{\mathrm{f}}$(C) denotes the participation predicate of VC$_{\mathrm{f}}$(C), where C $\in C_{base}$.
The functional state of an element of O is the set of functions in which o participates. it is denoted by EF(o) where o $\in O$.
$\forall$ o $\in O$, EF(o) = { f $\in F$ such as Pred$_{\mathrm{f}}$(class(o)) is verified for o}

<u>Extension of view classes</u>

extension(VC$_{\mathrm{f}}$(C)), where C $\in C_{base}$, is defined as follows:
extension(VC$_{\mathrm{f}}$(C)) = { o $\in$ extension(C) such as f $\in EF$(o)}

property 2: if C is a subclass of C' and f is a function, then extension(VC_f(C)) $\subseteq$ extension(VC_f(C'))
Let o be an element of extension(VC_f(C)), we know that f $\in EF$(o) and o $\in$ extension(C)
Since C is a subclass of C', o $\in$ extension(C') and we can prove that o $\in$ extension(VC_f(C'))
Property 3: if C is a subclass of C' and f is a function, then VC_f(C) is a subclass of VC_f(C')
This property is a direct consequence of properties 1 and 2.

4.3 Inheritance of the participation constraints

We know that an instance belonging to a view class belongs also to the extension of the superclasses (which are view classes) of this view class. This occurs even if this instance does not verify the participation predicate of these superclasses. This instance can violate this predicate, which is not very coherent. The solution to avoid this incoherence is to compute the participation predicate of a view class as the result of the conjunction of its own predicate and the predicate of all its superclasses. This conjunction guarantees that every instance belonging to a view class verifies the participation predicate of this view class, even instances of subclasses of this view class. We say that the participation predicate of a view class is inherited into its subclasses.

In view schemas, the inheritance relationship applies also for participation predicates, which reinforces its contextualization.

Example: in order to participate to the management function, a product must have a turnover greater than 50000 euros (see Figure 4). Professional products must verify this participation constraint due to the inheritance of the predicate of "Product" but they must also verify that their relative average margin is greater than 35% (i.e. averageMargin/salePrice$\geq$0.35). Consumer products must verify that their relative average margin is greater than 20%, the participation predicate of "Product" is also inherited into "ConsumerProduct" view class.

5 Contextualization of the inheritance relationships of the base schema

We focus now on the sharing of the inter-objects relationships by the view schemas. Within a view schema, the extremities of an inter-objects relationship defined in the base schema are the view classes whose base classes are the extremities of this relationship in the base schema. The extension of these view classes can be a subset of the extension of their base classes. This can pose a problem regarding the respect of referential integrity and cardinality constraints within the view schema. These constraints are associated with the semantics of the relationships and we want to share the relationships in the view schemas with their entire semantics. The differences between the extensions in the base schema and in one of the view schemas can result in a violation of the constraints in this view schema, while these constraints are still verified in the base schema. We will see that the solution to keep these constraints verified in view schemas will lead us to a contextualization of the inter-objects relationships.

5.1 Referential integrity constraints

Within a view schema sharing a relationship of the base schema, the extremities of some occurrences could not be present as a result of the differences of extensions between view classes and their base classes. This results in a violation of referential integrity constraints in this view schema.

Within a view schema sharing a relationship, the solution consists in using only occurrences whose extremities are present in this view schema. The sharing is then reduced by the extensions of the view classes located at the extremities of the shared relationships.

Example: the relationship of the base schema "purchaser/acquired" between "Customer" and "Product" is shared by the view schema corresponding to the management function. Figure 6 shows this relationship within the base schema with an example of occurrences of this relationship between instances of the classes "Customer" and "Product".

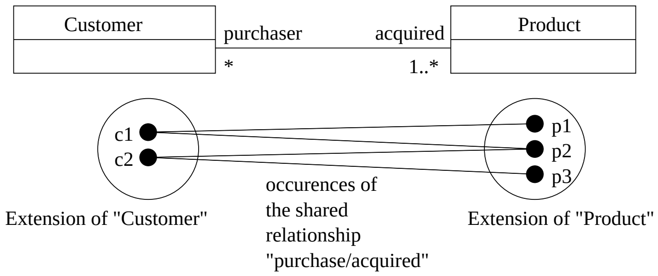

Fig. 6. Relationship "purchaser/acquired" introduced in the base schema

Figure 7 shows an example of the sharing of the relationship "purchaser/acquired" by the view schema corresponding to the management function. This view class "Product" introduces a participation constraint (see Figure 4) so the instance "p3" does not belong to its extension. Consequently, the occurrence between "c2" and "p3" is not used within the view schema. This guarantees that the referential integrity constraints are verified as shown on Figure 7.

Formalization

Respect of the referential integrity of shared relationships in view schemas

R_{base} denotes the set of inter-objects relationships introduced in the base schema.

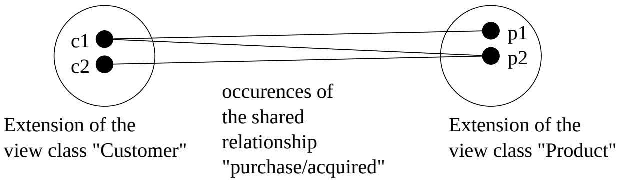

Fig. 7. Relationship "purchaser/acquired" and extensions of the view classes "Customer" and "Product" in the management function

Let R(C,C') be a relationship between C and C' classes of C_{base}.
occ$_{base}$(R(C,C')) denotes the set of occurences of R(C,C') in the base schema.
Referential integrity in the base schema is formulated as follows:
$\forall$ R $\in$ R_{base}, occ$_{base}$(R(C,C')) $\subseteq$ extension(C)$\times$extension(C') where (C,C') $\in$ C^2.
occ$_{\mathrm{f}}$(R(C,C')) denotes the set of occurences of R(C,C') in the view schema associated with the function f (f $\in$ F). This set is obtained as follows:
occ$_{\mathrm{f}}$(R(C,C'))= occ$_{base}$(R(C,C')) $\cap$ extension(VC$_{\mathrm{f}}$(C))$\times$extension(VC$_{\mathrm{f}}$(C'))
This set guarantees tha the referential integrity is verified in this view schema.

5.2 Cardinality constraints

In the same way as referential integrity constraints, the cardinality constraints can be verified in the base schema without being verified within one of the view schemas. The reason comes from the differences between the extension of the view classes and the extension of their base classes. More precisely, it is the minimal cardinality which cannot be verified. This results from the fact that only a subset of the occurrences of the relationship is used in the view schema in order to enforce the referential integrity.

If, at one time, the cardinality constraints of a relationship are verified within the base schema and not within one of the view schemas, we do not want to trigger an exception. In fact, the goal of relationship sharing is not to increase the constraints associated with the relationships but to use them with their complete semantics. Consequently, the only way to verify the minimal cardinality constraint of a shared relationship in every view schema is to exclude from it all instances that cause the violation of this constraint. This adds a new participation constraint to view shemas.

Example: we continue with the example of the relationship called "purchaser/acquired" introduced in the base schema between "Customer" and "Product" (see Figure 6). This relationship introduces a minimal cardinality of one for the instances of "Customer". So, the instances of "Customer" that do not verify this minimal cardinality of one in the management function do not participate to the corresponding view schema. This can occur since the management function introduces participation constraints for the instances of "Product" (see Figure 4).

Figure 8 shows this case for the instance "c2" of "Customer". There is no occurrence of "purchaser/acquired" that has "c2" as one of its extremities since the products "p2" and "p3" do not participate to the view schema corresponding to the management function. Consequently, "c2" does not participate to this view schema. The fact that such a customer does not participate the management function is justified by the semantics of a customer. This semantics contains the fact that a customer has purchased at least one product. A customer that did not purchase any product relevant to the management function is not considered as a customer relevant to this function.

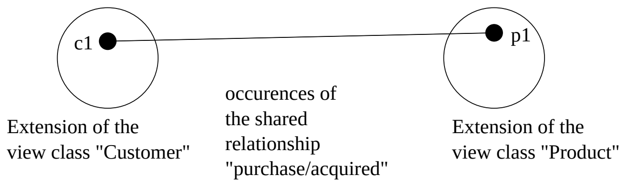

Fig. 8. Relationship "purchaser/acquired" and extensions of the view classes "Customer" and "Product" in the management function

The inter-objects relationships of the base schema own generic properties due to their contextualization in the view schemas. This contextualization is the result of their sharing which enforces the referential integrity and cardinality constraints in accordance with the extension of the view classes. Enforcing the referential integrity constraints leads to a restriction of the set of occurrences of a relationship. Enforcing the cardinality constraints creates a new participation constraint of the instances whose class is located at one of the extremities of the relationship.

Formalization

Respect of the minimal cardinality constraints of shared relationships in view schemas

We define, for the relationship R(C,C') $\in$ R$_{base}$((C,C') $\in C^2$), the following cardinalities:
card-min$_{C}$(R(C,C')) is the minimal cardinality of R(C,C') for the extremity C.
card-min$_{C'}$(R(C,C')) is the minimal cardinality of R(C,C') for the extremity C'.
Minimal cardinality constraints are verified in the base schema if and only if:
$\forall$ R(C,C') $\in R_{base}$ where (C,C') $\in C^2$,
$\forall$ o $\in$ extension(C),
card-min$_{C}$(R(C,C'))$\leq \|\{$(u,v) $\in$ occ$_{base}$(R(C,C')) such as u=o$\}\|$
$\forall$ o' $\in$ extension(C'),
card-min$_{C'}$(R(C,C')) $\leq \|\{$(u,v) $\in$ occ$_{base}$(R(C,C')) such as v=o'$\}\|$
The same constraints must be checked in each view schema, that is for every

schema view associated with the function f (f $\in F$):
$\forall$ R(C,C') $\in R_{base}$ where (C,C') $\in C^2$,
$\forall$ o $\in$ extension(VC_f(C)), card-min_C(R(C,C'))$\leq$ $\|\{$(u,v)$\in occ_f$(R(C,C')) such as u=o$\}\|$
$\forall$ o' $\in$ extension(VC_f(C')), card-$min_{C'}$(R(C,C'))$\leq$ $\|\{$(u,v)$\in occ_f$(R(C,C')) such as v=o'$\}\|$
These constraints represent a new participation constraint, leading to a new definition of the functional state:
$\forall$o$\in O, EF$(o)=$\{$ f $\in F$ such as:

- $Pred_f$(class(o)) is verified for o
- $\forall$ R(C,C') $\in R_{base}$ such as (C,C') $\in C^2$ and class(o) subclass of C, card-min_C(R(C,C'))$\leq$ $\|\{$ (u,v)$\in occ_f$(R(C,C')) such as u=o$\}\|$

$\}$

This definition has the drawback of using occ_f(R(C,C')) which is computed using the extension of view classes. We can easily rewrite this new definition of the functional state:
$\forall$o$\in O, EF$(o)=$\{$ f $\in F$ such as:

- $Pred_f$(class(o)) is verified for o
- $\forall$ R(C,C') $\in R_{base}$ such as (C,C') $\in C^2$ and class(o) subclass of C, card-min_C(R(C,C'))$\leq$ $\|\{$ (u,v)$\in occ_{base}$(R(C,C')) such as u=o and f $\in EF$(v)$\}\|$

$\}$

6 Comparison with related works

We have shown that CROME is a view approach which gives to the base schema generic properties against the view schemas. These properties are the result of the sharing and the contextualization of the relationships of the base schema within the view schemas (inheritance and inter-objects relationships).

In literature about view mechanisms ([2], [3], [4]), view classes are defined by the application of a query operator to one or two classes of the base schema. This definition takes into account this or these base classes to compute the view class but does not take into account the hierarchy of the base schema. This results in a fine granularity of view classes. The view schemas are an assembly of classes and view classes explicitly and freely chosen by the database administrator. In each view schema, the inheritance relationships are computed by a classifier ([5], [14]) based on the subsumption relationship of the model. Generally, the hierarchy computed by this classifier and the hierarchy of the base schema will not be isomorphic at all[4].

This proves that the inheritance relationships of the base schema are not generic and they are totally lost in the view schemas.

[4] Unless view classes has been defined for this purpose [7]

The second point concerns the preservation of the link between an instance and its class. In the base schema, each instance is described by its class which inherits the description of its superclasses. In CROME, within every view schema, this link is preserved and exists between an instance and the view class issued from its class. On the one hand, an instance is described only by this view class as a result of the isomorphism of the inheritance relationship between the base schema and the view schemas. On the other hand, this instance belongs to the extension of view classes issued from its class and to the extension of their superclasses. Moreover, this instance does not belong to the extension of any other class. This results from the choice for each instance of a unique participation constraint within each view schema.

In view mechanisms, the role of the class of an instance is not preserved at all. In a view schema, an instance can belong and be described by several view classes such that there exists no inheritance relationship between them. Indeed, the extension of each view class is computed by using the entire extension of its base class without taking into account the class of each instance which belongs to this extension.

Finally, the integrity constraints associated with the inter-objects relationships are not taken into account by view mechanisms. In a view schema, one of the extremities of an occurrence of a shared relationship can be an instance which does not participate to this view schema. This results from the fact that all occurrences of a relationship are shared without taking into account selections of instances. There exists only some ad-hoc techniques that use computed attributes in order to resolve the problems of referential integrity constraints[2].

7 Conclusion

We have proposed the CROME model in order to build view schemas which take into account the base schema at a level of granularity higher than in others works about object oriented views. This is justified by the richer semantics of the object model and in particular, by the inheritance and inter-objects relationships. In CROME, we have seen that these two relationships are shared by the whole set of view schemas. So, these view schemas are isomorphic to the base schema. Moreover, these relationships are contextualized by taking into account the local aspects of the view schemas. All this give a true genericity to the base schema as a graph of classes.

CROME has been implemented as a design tool[15] of the base schema and view schemas[5]. This tool can be used with any OODB system having a binding with Java. This tool has been adapted for two OODB systems: PSE/PSE pro[6] and POET[7]. This tool contains a design browser which takes into account the whole set of schemas (base and view schemas) of the database. This browser checks the coherence of the generated schemas to avoid some design mistakes.

[5] This tool is available on `http://www.lifl.fr/DOWNLOAD/`

[6] `http://www.objectdesign.com`, PSE is a registered mark of Object Design, Inc

[7] `http://www.poet.com`, POET is a registred mark of POET software corporation

The generation of the Java schemas uses the notion of interfaces and packages of this language.

References

1. ANSI/X3/SPARC. Study Group on Database Management Systems. In *ACM Sigmod*, 1975.
2. A. Geppert, S. Scherrer, and K. Dittrich. Derived Types ans Subschemas: Towards Better support for Logical Data Independence in Object-Oriented Data Models. Technical report, Institut fur Informatik., Universitat Zurich, 1993.
3. E.A. Rundensteiner. A Methodology for Supporting Multiple Views in Object-Oriented Databases. In *Proceedings of the 18th International Conference on Very Large Databases (VLDB'92)*, pages 187–198, August 1992.
4. M.H. Scholl, C. Laasch, and M. Tresch. Updatable Views in Object-Oriented Databases. In *Proceedings of the Deductive and Object-Oriented Databases, Second International Conference*, pages 189–205, December 1991.
5. A. Borgida, R. Brachman, D. McGuinness, and L.A. Resnick. CLASSIC: a Structural Data Model for Objects. *SIGMOD Record*, 18(2):59–67, 1989.
6. L. Debrauwer, B. Carré, and G. Vanwormhoudt. Un cadre de conception par contextes fonctionnels de systèmes d'information à objets. In *XVème Congrès INFORSID*, Toulouse, juin 1997.
7. L. Debrauwer. *Des vues aux contextes pour la structuration fonctionnelle de bases de données à objets en CROME.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille I, Lille, décembre 1998.
8. G. Vanwormhoudt, B. Carré, and L. Debrauwer. Programmation par objets et contextes fonctionnels. Application de CROME à Smalltalk. In *LMO'97*. R.Ducournau et S.Garlatti, octobre 1997.
9. G. Vanwormhoudt. *CROME : un cadre de programmation par objets structurés en contextes.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille I, Lille, septembre 1999.
10. E.A. Rundensteiner. A Transparent Object-Oriented Schema Change Using View Evolution. In *Proceedings of the IEEE international Conference on Data Engineering*, Taipei, March 1995.
11. L. Al-Jadir, G. Falquet, and M. Léonard. Context Versions in an Object-Oriented Model. In *Proceedings of the DEXA Conference*, 1993.
12. L. Al-Jadir, A. Le Grand, M. Léonard, and O. Parchet. Contribution to the Evolution of Information Systems. In T.W. Olle A.A. Verrijn-Stuart, editor, *Methods ans Associated Tools for the Information Systems Lifecycle,IFIP*. Elsevier, 1994.
13. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, 1999.
14. E.A. Rundensteiner. A Classification Algorithm for Supporting Multiple Views in Object-Oriented Databases. In *Proceedings of the ACM third International Conference on Information and Knowledge Management (CIKM'94)*, pages 18–25, November 1994.
15. O. Caron, B. Carré, and L. Debrauwer. CromeJava: une implémentation du modèle crome de conception par contextes pour les bases de données à objets en Java. In *Proceedings of LMO'00*, January 2000.