Normalization of Object-Oriented Conceptual Schemes

Eric ANDONOFF

Laboratoire IRIT pôle SIG,
Université Toulouse III, 118 route de Narbonne, 31062 Toulouse Cédex.

Abstract. This paper presents a normalization process for object-oriented conceptual schemes modelized with the O* object-oriented analysis method. Two normal forms called 1-ONF and 2-ONF and a synthesis process are described. These normal forms and synthesis process are used to validate only the structural part of a conceptual scheme; the behavioural part is out of the scope of the paper.

1 Introduction

Nowadays analysis methods of information systems (IS) are widely influenced by the object-oriented paradigm. This paradigm has already been used in several fields such as programming languages [15], design methods [8], artificial intelligence [22] or databases [9]. Analysis methods which integrate this paradigm are called object-oriented analysis method : they lead to the modeling of object-oriented conceptual schemes.

But those methods do not still deal with the normalization principle. They do not provide with solutions to indicate if the modelized conceptual schemes carry out desirable properties such as completeness or non redundancy. To determinate if a conceptual scheme is normalized, it is necessary to recognize the undesirable classes it describes and to convert them in a more desirable form.

We can note that such studies have been done in the relational context. Normal forms and normalization processes have been proposed to modelize suitable relational schemes [10], [7], [11]. These results have been used to validate the structural part of entity-relationship conceptual schemes [13].

This paper presents a normalization process adapted to the object-oriented conceptual schemes modelized with the object-oriented analysis method O* [9]. This method has been choosen because it fully integrates the object-oriented paradigm during the modeling stage [20]. Two normal forms called 1-ONF and 2-ONF and a synthesis process have been defined. The normal forms are used to check the redundancy of the conceptual scheme classes. The design process is used to convert undesirable classes in a more suitable form. The starting point of the process is a set of attributes, a set of functional (FD) and multivalued (MD) dependencies extended to the notion of role [3]. The result is a set of normalized classes. We can note that we approach only the structural part of the conceptual scheme; the approach of the behavioural part is out of the scope of this paper.

This paper is organized as follows. Section 2 is a survey of normalization processes for databases. Section 3 outlines the fundamental concepts of the O* model. Section 4 describes the normal forms and the normalization process. First some basic notions and notations are introduced. Then the normal forms and the design process are successively

presented. Section 5 illustrates the normalization principle of the structural part of O* object-oriented conceptual schemes using the previous design process. Finally some concluding remarks are given.

2 Survey of normalization processes for databases

On the one hand, normalization processes have been defined for relational databases. The starting point of these processes are a set of attributes and a set of FD or MD dependencies. Attributes describe the structural part of the database while dependencies express the semantics of the database. The result is a set of normalized relations for which redundancy is reduced : 3NF, BCNF, 4NF ...

The most famous processes are the synthesis and the decomposition algorithms. The synthesis algorithm described in [7] modelizes, from a set of attributes and a set of FD, 3NF relational schemes. The decomposition algorithm proposed in [11] lead to 4NF relational schemes from a set of attributes and a set of MD. An algorithm which combines the synthesizing and the decomposition approaches is presented in [6]. This algorithm considers a set of attributes and a set of FD and MD to modelize 4NF relational schemes. The underlaying process first uses the decomposition approach to determine a set of clusters (set of attributes and dependencies) from the MD; then the synthesis approach is used to deal the clusters with respect to the set of FD.

On the other hand, extended relational databases [1] [18] have been proposed to make the modelization of strongly structured data easier. Such databases consider that tuple components of relations may be sets, lists or relations themselves; they are free from 1NF. Normalization processes for such databases have been proposed. First, [14] extends the notion of FD and MD for nested relations and defines a normal form integrating those extended dependencies in the definition of the 4NF. Then, [21] introduces the PNF (Partitioned Normal Form) which represents scheme of nested relations by trees. Finally, [16], [17] take its inspiration from the previous works to define the NNF (Nested Normal Form).

3 The O* object-oriented model

O* [9] is an object-oriented analysis method which consists of an object-oriented model which fully integrates the object-oriented paradigm [20] and a set of methodological guidelines. For reasons of space limitation, this section only describes the O* model.

The O* model provides with two kinds of schemes to describe the static and dynamic aspects of an IS. These schemes integrate the concepts of object and classe.

The static scheme describes the static aspects of an IS through classes connected by inheritance links and attributes defined on classes. These attributes represent the static links modelized between classes. Two kinds of links are distinguished : the composition links and the refering links. A composition link between a composite class and a component class expresses a strong coupling of behaviour between a composite object and its component object: on one hand, the existence of the component object depends on the existence of its composite object; on the other hand, a component object belongs to one and only one composite object. In contrast, a refering link between a refering class and a

refered class expresses a weak coupling of behaviour between a refering object and its refered object: on one hand, the existence of the refered object is totally independent of its refering object; on the other hand, a refered object may be shared by several refering object. Theses links are either simple or multiple; simple links indicate that a composite (a refering) object is composed of (refers to) one and only one object while multiple links indicate that a composite (a refering) object is composed of (refers to) one or several objects. An example of static scheme is given in section 4.

The dynamic scheme describes the behavioural aspects of the IS using the concepts of events and operations associated with classes. Events may be external, internal and temporal. External events correspond to the events occurring in the environment outside an IS. Internal events correspond to the internal state changes or rather the system answers of the IS. Temporal events correspond to the events whose occurrences depend on the description of time. On the other hand, an operation represents an action performed on an object of a class and causes a state change to this object.

4 Normal forms and normalization process

Our proposal for the normalization of O* object-oriented scheme takes its inspiration from the normalization principles of extended relational databases. It re-uses the notion of tree to represent the classes structure describing the modelized conceptual scheme and leads to a structural normalization.

4.1 Preliminaries

We use the notions of FD [10], MD [11] and role associated with a dependency:

- There is a FD between X and Y, noted $X \rightarrow Y$, if to each value of X corresponds one value of Y at the most in a O* class C(X,Y,Z).

- There is a MD between X and Y, noted $X \twoheadrightarrow Y$, if to each value of X corresponds one or several values of Y, independent of the values of Z in a O* class C(X,Y,Z).

- The notion of role precises the semantics of dependencies allowing the distinction between identical dependencies (i.e. expressed between the same left and right hand). The role is noted between brackets. It allows the modelization of several relationships between the same classes; it is also used to traduce the inheritance semantic expressed by a FD (see section 4).

We also use the different inference rules defined for FD and MD [4]. We require these rules:

- To compute the dependents of a set X of attributes: those dependents correspond to all the attributes Y which are inferred from X and from a set of FD and MD, applying the previous inference rules [5].

- To compute the transitive closure and a minimal cover of a set D of FD and MD: the transitive closure of D, noted D^+, is the set of dependencies which may be inferred from D; a minimal cover of D, noted D^-, is a set of reduced (i.e. exempt of redundancy) and non inferred dependencies (from D).

A dependency d:X— W ∈ $(D^-)^+$, (where — indicates the functional or multivalued feature of the dependency), is reduced if it is [16] :

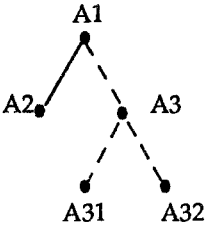
non trivial	if XW = U	then W≠∅ et W ⊄ X
left reduced	∀ X', X'⊂ X	then X'— W ∉ $(D^-)^+$
right reduced	∀ W', W'⊂ W	then X— W' ∉ $(D^-)^+$ where X— W is trivial
non transferable	∀ X', X'⊂ X	then X'— W(X-X') ∉ $(D^-)^+$

- D^- and D^+ allow to determinate the keys of D. Those keys are either essential or nonessential. The essential keys of D are the left-hand of D^- dependencies . The nonessential keys of D are the left-hand of D^+ reduced dependencies.

Finally, we use the notion of tree to represent conceptual scheme classe structure. A tree is described by a set of nodes and leaves which correspond to the attributes of the classes, and by a set of edges which express the FD and MD existing between the attributes. O* references are represented by a leaf of a tree $\mathcal{A}'$ which is also the root of a tree $\mathcal{A}''$. O* compositions correspond to sub-trees of any tree.

The following tree $\mathcal{A}$ illustrates the different structural integrity constraints which exist between its attributes. It represents the following dependencies :

A1 → A2
A1 →→ A3A31A32
A1A3 →→ A31
A1A3 →→ A32



where FD and MD are respectively noted —— and – – – on the trees. We note DPS($\mathcal{A}$) the set of dependencies represented by the tree $\mathcal{A}$.

4.2 Normal forms

To validate the previous trees, we need to define normal forms. We present here two normal forms called 1-ONF and 2-ONF. 1-ONF checks if a tree corresponds to the set of FD and MD it must describe while 2-ONF verifies if a 1-ONF tree is non redundant.

☞ **1-ONF**

Let D be the set of FD and MD, D^- a minimal cover of D and $\mathcal{A}$ the corresponding tree. $\mathcal{A}$ is in 1-ONF if and only if every leaf is decomposed with respect to the essential keys of D. The notion of decomposed leaves is defined as follows. X , leaf of $\mathcal{A}$, is not decomposed:

- if it exists C such as X∩C≠∅ where C is a reduced essential key of D i.e. a key such as it does not exist C', an essential key of D, checking X∩C' ⊂ X∩C et X∩C'≠∅,

- $X=X'Y_1...Y_n$ with $X'=X \cap C$ and $Y_1...Y_n \neq \emptyset$, $Y_1 \in$ Dependent(A(P(X))X'), ..., $Y_n \in$ Dependent (A(P(X))X').

The idea is to use the notion of dependent to decompose every leaf. For example, let U={A, B, C, D, E, F, G} and D={A →→ B, B → C, AD →→ FG, AD →→ E}.

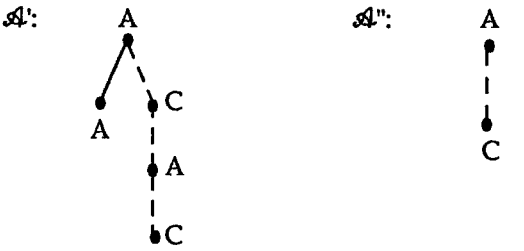
The following $\mathcal{A}'$ tree is not decomposed with respect to the essential keys {A, B, AD} because the leaf DEFG is not decomposed. Indeed, Dependent(AD)={A, D, {A}, {D}, {B}, {C}, {FG}, {E}}. Therefore, it exists X'=D, Y'=E and Y"=FG such as $Y' \in$ Dependent(AD) et $Y'' \in$ Dependent(AD). Quite the opposite, the $\mathcal{A}''$ tree is fully decomposed.

☞ **2-ONF**

Let D be the set of FD and MD, D^- a minimal cover of D and $\mathcal{A}$ the corresponding tree. $\mathcal{A}$ is in 2-ONF if and only if $\mathcal{A}$ is in 1-ONF and if $\mathcal{A}$ is nonredundant. Three kinds of redundancy are specified: reflexive, augmentative and transitive redundancy.

- **Reflexive redundancy**

This redundancy is illustrated through the example of the following tree $\mathcal{A}'$. Indeed DPS($\mathcal{A}'$)={A → A, AC →→ AC, ACA →→ A}. The dependencies A → A and AC →→ AC are reflexive redundant with respect to A and AC. To remove this redundancy, the edges b'=(A,A) and b"=(C,A) must be deleted. The result is described in $\mathcal{A}''$.



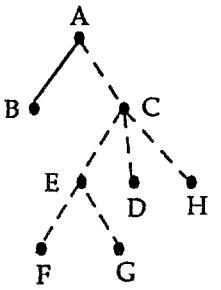
- **Augmentative redundancy:** two cases are possible.

Case 1 [17]:

Let U={A, B, C, D, E, F, G, H} and D={A → B, AC →→ EFG, BE →→ AF, BE →→ D}. D is a minimal cover and {A, AC, BE, AE, AEC, BEC} are the keys of D.

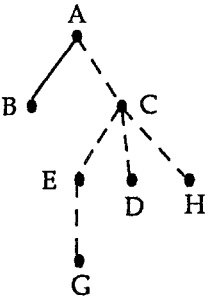
The $\mathcal{A}'$ tree is augmentative redundant. Indeed DPS($\mathcal{A}'$)={A $\rightarrow$ B, A $\rightarrow\!\!\!\rightarrow$ CEFGDH, AC $\rightarrow\!\!\!\rightarrow$ EFG, AC $\rightarrow\!\!\!\rightarrow$ D, AC $\rightarrow\!\!\!\rightarrow$ H, ACE $\rightarrow\!\!\!\rightarrow$ F, ACE $\rightarrow\!\!\!\rightarrow$ G}. Now AE is a key of D such as Dependent(AE)={A, E, {A}, {E}, B, {D}, {CGH}, {F}}. Therefore AE $\rightarrow\!\!\!\rightarrow$ F; so ACE $\rightarrow\!\!\!\rightarrow$ F is augmentative redundant with respect to AE.

$\mathcal{A}'$:

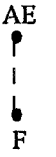


To remove this redundancy, the edge b=(E,F) of $\mathcal{A}'$ must be deleted and a new tree representative of the dependency AE $\rightarrow\!\!\!\rightarrow$ F must be created. Then the trees $\mathcal{A}'$ and $\mathcal{A}''$ are the following:

$\mathcal{A}'$:



$\mathcal{A}''$:

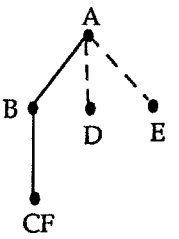


Case 2:

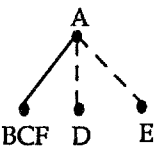
Let U={A, B, C, D, E, F} and D={A $\rightarrow$ BCF, A $\rightarrow\!\!\!\rightarrow$ D, BD $\rightarrow\!\!\!\rightarrow$ A}. D is a minimal cover of itself and {A, BD} are the keys of D.

The following tree $\mathcal{A}$ is augmentative redundant. Indeed DPS($\mathcal{A}$)={A $\rightarrow$ BCF, AB $\rightarrow$ CF, A $\rightarrow\!\!\!\rightarrow$ D, A $\rightarrow\!\!\!\rightarrow$ E}. As A $\rightarrow$ BCF implies AB $\rightarrow$ CF, AB $\rightarrow$ CF is augmentative redundant with respect to AB. To remove this redundancy, the leaf AC must be gather with the node B. The tree $\mathcal{A}$ becomes:

$\mathcal{A}$ (1):



$\mathcal{A}$ (2):



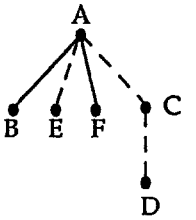
- **Transitive redundancy:** two cases are possible.

Case 1:

Let U={A, B, C, D, E, F} and D={A→B, B→→E, B→F, C→→D}. D is a minimal cover of itself and {A, B, C} are the keys of D.

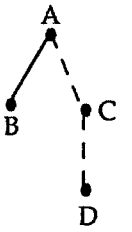
The following tree $\mathcal{A}'$ is transitive redundant. Indeed DPS($\mathcal{A}'$)={A→B, A→→E, A→F, A→→CD, AC→→D, ACE→→F, ACE→→G}. Now with respect to D, we have B→→E, B→F. Therefore, as A→B ∈ DPS($\mathcal{A}'$), A→→E and A→F hold. These dependencies are transitive redundant with respect to B.

$\mathcal{A}'$:



To remove this redundancy, the edges b'=(A,E) and b"=(A,F) of $\mathcal{A}'$ must be deleted and a new tree A" representative of the dependencies B→→E and B→F must be created.

$\mathcal{A}'$:

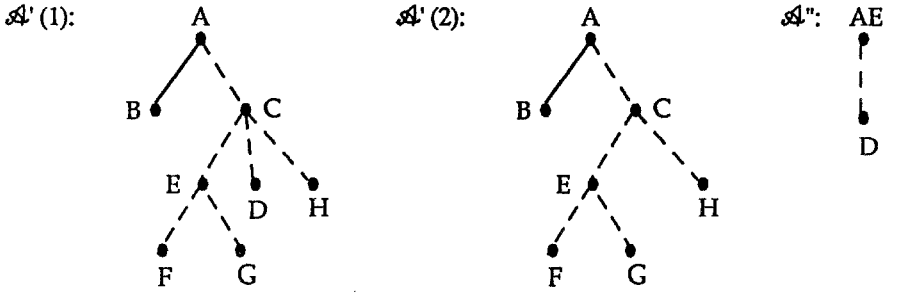


$\mathcal{A}$":

Case 2 [17]:

Let U={A, B, C, D, E, F, G, H} and D={A→B, AC→→EFG, BE→AF, BE→→D}. D is a minimal cover of itself and {A, AC, BE, AE, AEC, BEC} are the keys of D.

The following $\mathcal{A}'$ tree is transitive redundant. Indeed DPS($\mathcal{A}'$)={A→B, A→→CEFGDH, AC→→EFG, AC→→D, AC→→H, ACE→→F, ACE→→G}. Now AE is a key of D such as Dependent(AE)={A, E, {A}, {E}, B, {D}, {CGH}, {F}}. Therefore AE→→D. More over AC→→EFG implies AC→→AEFG and AE→→D implies AEFG→→D. Therefore AC→→D is transitive redundant with respect to AE. To remove this redundancy, the edge b=(C,D) of $\mathcal{A}'$ must be deleted and a new tree $\mathcal{A}$" must be created such as AE→→D.



4.3 Design process

The design process is a synthesis algorithm which modelize a set of trees which are directly in 2-ONF from a set of attributes and a set of FD and MD. This algorithm takes its inspiration from those proposed in the relational databases context [7]. It re-uses the notion of clusters introduced in [6]. It lies upon the basis inference rules (R1 ... R9) complemented by the rules R10, R11, R12 and R13:

R1 (Reflexivity)	if $Y \subseteq X$	then $X \rightarrow Y$
R2 (Augmentation)	if $Z \subseteq W$ and $X \rightarrow Y$	then $XW \rightarrow YZ$
R3 (Transitivity)	if $X \rightarrow Y$ and $Y \rightarrow Z$	then $X \rightarrow Z$
R4 (Complementation)	if $X \twoheadrightarrow Y$	then $X \twoheadrightarrow U - XY$
R5 (Reflexivity)	if $Y \subseteq X$	then $X \twoheadrightarrow Y$
R6 (Augmentation)	if $Z \subseteq W$ and $X \twoheadrightarrow Y$	then $XW \twoheadrightarrow YZ$
R7 (Transitivity)	if $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$	then $X \twoheadrightarrow Z - Y$
R8 (FD is a MD)	if $X \rightarrow Y$	then $X \twoheadrightarrow Y$
R9 (Projectability)	if $X \twoheadrightarrow Y$ and $Y \rightarrow Z$	then $X \rightarrow Z\text{-}Y$

R10 (Augmentation case 1)	if $X \rightarrow Y$ then $XY' \rightarrow Y''$ with $Y'Y''=Y$ and $Y' \neq \emptyset$ and $Y'' \neq \emptyset$
R11 (Augmentation case 2)	if $X \rightarrow Y$ (resp $X \twoheadrightarrow Y$) then $XZ \rightarrow Y$ (resp $XZ \twoheadrightarrow Y$)
R12 (Transitivity case 1)	rules R3, R5 and R8 are jointly used
R13 (Transitivity case 2)	if $X \twoheadrightarrow Y$ and $Z \twoheadrightarrow W$ with $Z=X'Y'$, $X' \subseteq X$ and $Y' \subseteq Y$ then $X \twoheadrightarrow W$

The last rules R10, ..., R13 correspond to the different cases of redundancy presented before; they can be considered as macro inference rules inferred from the basis rules.

We consider a formal example to illustrate the design process. Let U={A, B, C, D, E, F, G, H, I, J} and D={$A \rightarrow B$, $B \twoheadrightarrow E$, $B \rightarrow F$, $AC \twoheadrightarrow DJH$, $AJ \twoheadrightarrow H$, $AD \twoheadrightarrow J$, $AD \twoheadrightarrow G$}. The synthesis algorithm used is the following:

```
Compute a minimal cover of D
Compute the clusters (U',D') of D
For each cluster Do
        Compute the keys of D'
        Compute the dependents of the essential keys of D'
        Add in D' the inferred dependencies
        Use the rules R10, R11, R12 and R13 to remove the inferred dependencies
              obtained from D'
        Compute the dependents of nonessential keys of D'
        For each dependency d Do
              Use the rules R10, R11, R12 and R13 to replace the inferred
                    dependencies from D'+d by d (in D')
        End For
End For
D  <--  ∪ D'
Determine the 2-OFM corresponding trees
```

First, the design process computes a minimal cover of D to remove the redundancy of the non strongly structured data. In the example, D is a minimal cover of itself.

Then, the design process removes the redundancy of strongly structured data. It begins computing the different clusters of U with respect to D and the keys of these clusters. The clusters are groups of attributes, FD and MD expressed between these attributes. They are identified as follows:

```
Compute the CE set of essential keys of D
Arrange CE in growing order of keys (the order of a key is the number of
    attributes which compose it [2])
Select the keys whose order is equal and minimum
Make a cluster for each minimum order key; this cluster is defined as follows:
    D' = {d∈D such as LeftHand(d)∩(key of the cluster)≠∅}
    U' = {A such as A appear in d'∈D'}
Add the attributes of U which do not belong to any U' if one of the U'
Gather the equivalent clusters (two clusters are equivalent if their keys are such as
    (key1 →► key2)∈D and (key2 →► key1)∈D)
```

In the example, the clusters are the following:

- Cl1=(U1,D1) where U1={A, B, C, D, H, I, J} and D1={A →► B, AC →►► DJH, AJ →►► H, AD →►► J, AD →►► G}.
- Cl2=(U2,D2) where U2={B, E, F} and D2={B →►► E, B →► F}.

Now, the redundancy of the clusters must be removed. For each cluster, the designer first determinates the dependents of the essential keys of the cluster (using the principle decomposition of leaves presented in the 1-ONF) and add in D' the inferred dependencies. This step is illustrated through the example as follows:

- Cl1: the essential keys are A, AC, AD, AJ and the unique nonessential key is ACJ. The designer uses the notion of reduced keys to compute the dependent of essential keys; the computing principle is the same as making 1-ONF trees:

  Dependent(A)={A, {A}, B, {CDGHIJ}} implies A $\rightarrow$ B and A $\rightarrow\!\!\!\rightarrow$ CDGHIJ.
  Dependent(AC)={A, C, {A}, {C}, B, {DJH}, {G}, {I}} implies AC $\rightarrow$ B, AC $\rightarrow\!\!\!\rightarrow$ DJH, AC $\rightarrow\!\!\!\rightarrow$ G and AC $\rightarrow\!\!\!\rightarrow$ I.
  Dependent(ACD)={A, C, D, {A}, {C}, {D}, B, {G}, {H}, {I}, {J}) implies ACD $\rightarrow$ B, ACD $\rightarrow\!\!\!\rightarrow$ G, ACD $\rightarrow\!\!\!\rightarrow$ H, ACD $\rightarrow\!\!\!\rightarrow$ I and ACD $\rightarrow\!\!\!\rightarrow$ J.

  Therefore D1={A $\rightarrow$ B, AC $\rightarrow\!\!\!\rightarrow$ DJH, AJ $\rightarrow\!\!\!\rightarrow$ H, AD $\rightarrow\!\!\!\rightarrow$ J, AD $\rightarrow\!\!\!\rightarrow$ G, A $\rightarrow\!\!\!\rightarrow$ CDGHIJ, AC $\rightarrow$ B, AC $\rightarrow\!\!\!\rightarrow$ G, AC $\rightarrow\!\!\!\rightarrow$ I, ACD $\rightarrow$ B, ACD $\rightarrow\!\!\!\rightarrow$ G, ACD $\rightarrow\!\!\!\rightarrow$ H, ACD $\rightarrow\!\!\!\rightarrow$ I, ACD $\rightarrow\!\!\!\rightarrow$ J}.

  The dependencies AC $\rightarrow$ B, ACD $\rightarrow$ B, ACD $\rightarrow\!\!\!\rightarrow$ G, ACD $\rightarrow\!\!\!\rightarrow$ I and ACD $\rightarrow\!\!\!\rightarrow$ J are deleted because they are augmentative redundant (rules R10 and R11). Therefore D1={A $\rightarrow$ B, AC $\rightarrow\!\!\!\rightarrow$ DH, AJ $\rightarrow\!\!\!\rightarrow$ H, AD $\rightarrow\!\!\!\rightarrow$ J, AD $\rightarrow\!\!\!\rightarrow$ G, A $\rightarrow\!\!\!\rightarrow$ CDGIH, AC $\rightarrow\!\!\!\rightarrow$ G, AC $\rightarrow\!\!\!\rightarrow$ I, ACD $\rightarrow\!\!\!\rightarrow$ H}.
  The dependency AC $\rightarrow\!\!\!\rightarrow$ G is removed because it is transitive redundant with respect to AD (rule R12 applied from AC $\rightarrow\!\!\!\rightarrow$ DH and AD $\rightarrow\!\!\!\rightarrow$ G). The dependency ACD $\rightarrow\!\!\!\rightarrow$ H is also removed because it is transitive redundant with respect to AJ (rule R12 applied from ACD $\rightarrow\!\!\!\rightarrow$ J and AJ $\rightarrow\!\!\!\rightarrow$ H).

  Hence D1={A $\rightarrow$ B, AC $\rightarrow\!\!\!\rightarrow$ D, AJ $\rightarrow\!\!\!\rightarrow$ H, AD $\rightarrow\!\!\!\rightarrow$ J, AD $\rightarrow\!\!\!\rightarrow$ G, A $\rightarrow\!\!\!\rightarrow$ CDI, AC $\rightarrow\!\!\!\rightarrow$ I}.

- Cl2: the essential key is B. Computing its dependents doesé not alter D2 which has no redundancy.

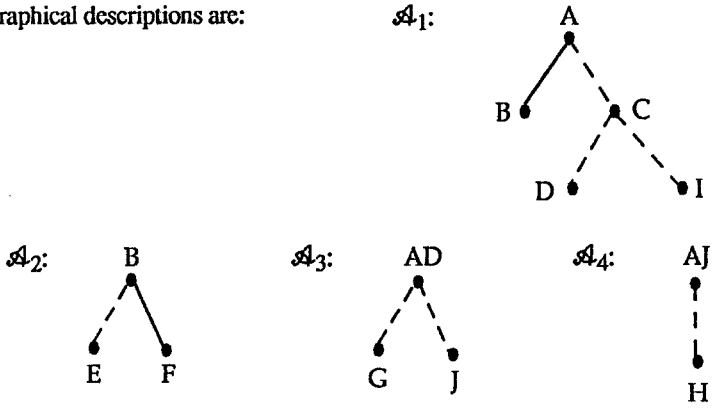
Then, the designer computes the dependents of the nonessential keys of the clusters. For each inferred dependency d, he checks if it exists in D' a dependency d' which is redundant with respect to Left-Hand(d) (which is a non essential key) in D'+d and replaces d' with d; rules R10, R11, R12 and R13 are applied. In the example, Cl1 has a nonessential key ACJ. Dependent(ACJ)={A, C, J, {A}, {C}, {J}, B, {D}, {H}, {G}, {I}}. The inferred dependencies are nonredundant in D1, so D1 is not altered.

When each cluster is considered, D is the union of each D'. In the example, D={A $\rightarrow$ B, B $\rightarrow\!\!\!\rightarrow$ E, B $\rightarrow$ F, AC $\rightarrow\!\!\!\rightarrow$ D, AJ $\rightarrow\!\!\!\rightarrow$ H, AD $\rightarrow\!\!\!\rightarrow$ J, AD $\rightarrow\!\!\!\rightarrow$ G, A $\rightarrow\!\!\!\rightarrow$ CDI, AC $\rightarrow\!\!\!\rightarrow$ I}.

Finally, the designer deduces the 2-ONF corresponding trees: he gathers together the dependencies which have the same left-hand. Two dependencies d:X—Y and d':X'—Y' have same left-hands if and only if X=X' or X'=XX" with X"$\subset$Y and Y'$\subset$Y. Gathered dependencies are the 2-ONF modelized trees. In the example, the trees are the following:

- $\mathcal{A}_1$=(U1,D1) with U1={A, B, C, D, I} and D1={A $\rightarrow$ B, A $\rightarrow\!\!\!\rightarrow$ CDI, AC $\rightarrow\!\!\!\rightarrow$ I, AC $\rightarrow\!\!\!\rightarrow$ D},
- $\mathcal{A}_2$=(U2,D2) with U2={B, E, F} and D2={B $\rightarrow\!\!\!\rightarrow$ E, B $\rightarrow$ F},
- $\mathcal{A}_3$=(U3,D3) with U3={A, D, J, G} and D3={AD $\rightarrow\!\!\!\rightarrow$ J, AD $\rightarrow\!\!\!\rightarrow$ G},
- $\mathcal{A}_4$=(U4,D4) with U4={A, J, H} and D4={AJ $\rightarrow\!\!\!\rightarrow$ H}.

Their graphical descriptions are:

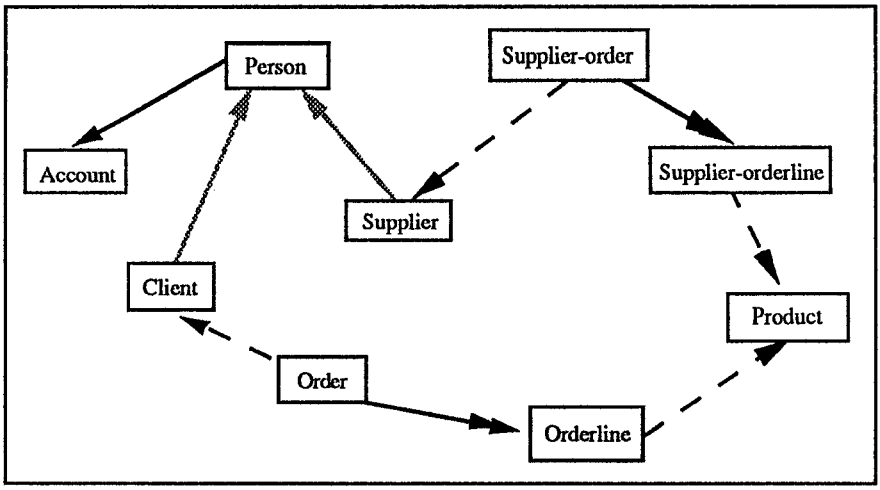


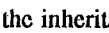
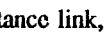
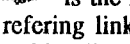
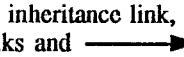
5 Normalization of O* object-oriented conceptual schemes

In this section, we describe how the previous normal forms may be used to normalize the static scheme modelized with the O* method.

The static scheme may be presented in a graphical way or a textual way. The two descriptions are useful to check if the static part of the conceptual scheme is normalized. The graphical description shows the different classes and the links which connect them. Three groups of links are illustrated: inheritance links, refering links and composition links. Refering and composition links may be simple or multiple. The textual description indicates the different attributes of each class of the graphical description.

Examples of graphical and textual static conceptual scheme are described hereunder. They help us to explain how we check the normalization of the static part of the conceptual scheme. In this example, we consider two main functions of a business firm namely order processing and inventory management [9]. The graphical description is the following:



where  is the inheritance link,  and  are the simple and multiple refering links and  and  are the simple and multiple composition links. The partial corresponding textual static conceptual scheme is the following:

```
class Order
compositions
   order#          : integer
   creation date   : string
   delivery date   : string
   invoice date    : string
   state           : string
   lines           : set-of (Orderline)
references
   client          : Client

class Orderline
compositions
   quantity        : real
references
   product         : Product
```

To check the normalization of such a conceptual scheme, we proceed as follows. First, the clusters are deduced from the graphical description; they correspond to the set of classes connected at least by a composition link. Next, the attributes of the clusters are recorded. These attributes are attributes which do not express a (refering or composition) link; they come from the textual description. Then, the cluster keys are choosen; they correspond to one or several attributes which identify the cluster. Next, the dependencies of the clusters are expressed. These dependencies traduce either existing links between classes of the conceptual scheme or existing links between the key of the cluster and the other attributes. Inheritance links are represented through FD and an inheritance role. Simple refering or composition links correspond to FD. But there is not always a close correspondence between a multiple link and a MD because of the strong restriction of MD (there is a MD between X and Y if to each value of X corresponds one or several values of Y independent of the values of Z).

In the business firm example, six clusters are identified. The first one gathers the Person and Account classes, the second the Client class, the third the Order and Orderline classes, the fourth the Supplier class, the fifth the Supplier-order and Supplier-orderline classes and the sixth the Product class.

The attributes of these clusters are the attributes of the classes they gather except those which describe links coming from these classes. For example, the attribute *client* of the Order class is not keeped in the third cluster (Order and Orderline).

The keys of these clusters are the attributes which identify the composite classes. For example, the key of the third cluster is the attribute *order#* of the Order class : such an attribute is called conceptual identifier in [19].

The dependencies of these clusters describe links between their keys and the other attributes. For example, *order#* → *state* and *order#* → *creation-date* are dependencies of the third cluster. These dependencies can also describe inheritance, refering or composition links. The third cluster contains the dependency *order#* → *client#* which represents a refering link between the Order class and the Client class. It also contains the dependencies

order# →→ orderline#, order# orderline# → product# and *order# orderline# → quantity.* The inheritance link existing between the class Client and the class Person is represented by a FD between the corresponding clusters. An inheritance role is associated to this FD [3]: *client# → pers#(inheritance).*

Then, a minimal cover is computed for each cluster; the design process presented in section 4.3 is also used to remove redundancy of strongly structured data in each cluster and to deduce the corresponding set of 2-ONF trees. Finally, the static O* scheme is built again:

- trees and sub-trees correspond to classes;
- nodes and leaves correspond to attributes;
- sub-trees express composition links;
- leaves which are also root of a tree express refering links except if there is an inheritance role associated with the considered leaf; in that case, they express an inheritance link.

The obtained static O* scheme is normalized with respect to the 2-ONF.

6 CONCLUSION

This paper has presented a normalization process adapted to the object-oriented conceptual schemes modelized with the object-oriented analysis method O* [9]. This method has been choosen because it fully integrates the object-oriented paradigm during the modeling stage [20]. Two normal forms called 1-ONF and 2-ONF and a synthesis process have been proposed. The normal forms are used to check the redundancy of the conceptual scheme classes and the design process is used to convert undesirable classes in a more suitable form. The starting point of the process is a set of attributes, a set of functional and multivalued dependencies extended to the notion of role [3]. The result is a set of normalized classes. We think that such a process is useful in object-oriented design methods in order to support the modelization of "well-defined" conceptual schemes.

We can note that only the structural part of the conceptual scheme is aproached; the approach of the behavioural part is out of the scope of this paper. Our current works concern this behavioural part; the problem may be dealt minimizing the complexity of the client/server graph.

References

1. S. Abiteboul: Non-first normal form relations: an algebra allowing data structuring. Journal of Computer and System Science, December 1986.
2. E. Andonoff, C. Sallaberry, G. Zurfluh: Interactive design of object-oriented databases. 4th CAiSE International Conference, Manchester, May 1992.
3. E. Andonoff: OFM: une méthode formelle pour la conception de bases de données orientées objet. Thèse de l'Université de Toulouse III, Septembre 1992.
4. C. Beeri, R. Fagin, T. Howard: A complete axiomatization for functional and multivalued dependencies. 2nd ACM International Conference on Management of Data, Toronto, August 1977.
5. C. Beeri: On the membership problem for functional and multivalued dependencies in relational database. ACM Transaction On Database Systems, Vol. 5, n°3, September 1980.

6. C. Beeri, M. Kifer: An integrated approach for logical relational design of relational database schemes. ACM Transaction On Database Systems, Vol. 11, n°2, June 1986.

7. P. Bernstein: Synthesizing third normal form relations from functional dependencies. ACM Transaction On Database Systems, Vol. 1, n°4, December 1976.

8. G. Booch: Software engineering with ADA. Benjamin-Cummings publishing company, 1991.

9. J. Brunet: Modelling the world with semantic objects. IFIP TC8 Working Conference on Object-Oriented approach in Information Systems, Quebec, October 1991.

10. E. Codd: Further normalization of the database relational model. In Database systems, Prentice-Hall publishing company, Englewood Cliffs, 1972.

11. R. Fagin: Multivalued dependencies and new normal form for relational databases. ACM Transaction On Database Systems, Vol. 2, n°3, September 1977.

12. W. Kim: Object-oriented databases: definition and research directions. IEEE Transaction on Knowledge and Data Engineering, Vol. 2, n° 3, September 1990.

13. T. Ling: A normal form for entity relationship diagrams. 4th ER Approch International Conference, Chicago, November 1985.

14. A. Makinouchi: A consideration on normal form of not-necessarily-normalized relations in the relational data model. 3rd VLDB International Conference, Tokyo, 1977.

15. B. Meyer: Object-oriented software construction. Prentice-Hall publshing company, Englewood Cliffs, 1988.

16. Z. Ozsoyoglu, L.Y. Yuan: A normal form for nested relations. 4th ACM Symposium on Principles of Database Systems, March 1985.

17. Z. Ozsoyoglu, L.Y. Yuan: A new normal form for nested relations. ACM Transaction On Database Systems, Vol. 12, n°1, 1987.

18. P. Pistor, F. Andersen: Designing a generalized NF2 model with an SQL-type language interface. 12th VLDB International Conference, Kyoto, 1986.

19. C. Rolland. C. Cauvet: Modélisation conceptuelle orientée objet 7ièmes Journées Bases de Données Avançées, Lyon, Septembre 1991.

20. C. Rolland: Trends and perspectives in conceptual modeling. Indo-French Workshop on Object-Oriented Databases, Goa, November 1992.

21. M. Roth, H. Forth, A. Silberschatz: Theory on non-first-normal form relational databases. Internal Report 84/36, Univeristy of Texas, Austin, December 1984.

22. M. Stefik, D. Bobrow: Object-oriented programming: themes and variations. Articial Intelligence Magazine, January 1986.