DRIVER II (Lisp)
Manuel de Référence

Franck Lebastard

Janvier 1997

Rapport CERMICS N^o 97-87

DRIVER II (Lisp)
Manuel de Référence

Franck Lebastard
Équipe « Base de données » - CERMICS
2004 route des Lucioles BP93
06902 Sophia Antipolis Cedex - France

e-mail : Franck.Lebastard@sophia.inria.fr
http://www.inria.fr/cermics/dbteam/Franck.Lebastard/

Résumé

Ce document constitue la documentation du système de gestion de bases de données multimodèles DRIVER II version LISP.

DRIVER II définit un SGBD multi-modèles au dessus d'un SGBD relationnel. Une base de données relationnelle est accessible sous forme relationnelle, NF2 ou objet après définition d'un schéma de correspondances décrivant les représentations de chaque modèle à rapprocher et les correspondances concrètes entre elles. Un des atouts du système DRIVER est qu'il permet à chaque utilisateur de choisir le modèle et les structures dans lesquels vont lui être présentées les données.

Abstract

This document is the reference manual of the multimodel database management system DRIVER II for the LISP version.

DRIVER II defines a multimodel DBMS on the top of a relational DBMS. A relational database can be accessible as a relational (views), an NF2 or an object database after that a correspondence schema describing model representations and concrete mapping between them has been defined. One of the main interests of the DRIVER system is that each user can choose the model and the structures in which he sees and handles (shared) data.

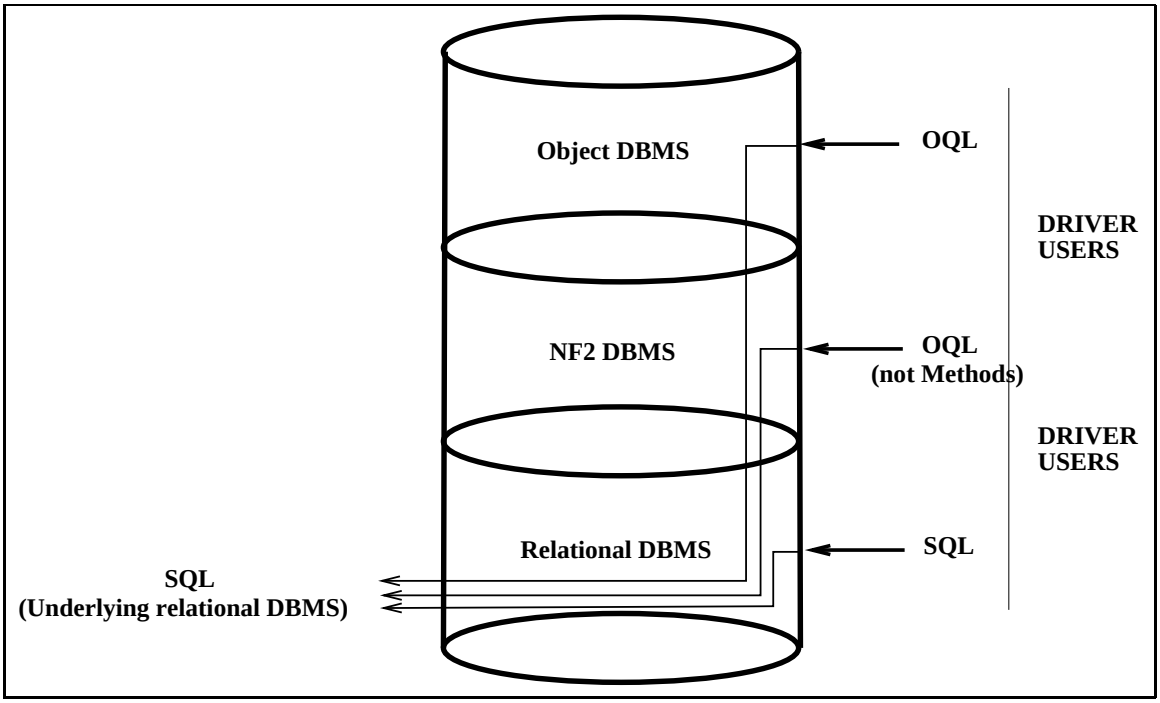

FIG. 1 – *L'architecture générale de Driver II*

DRIVER **[FEATURE]**

Ce trait indique que le module DRIVER est chargé en mémoire. Si ce n'est pas le cas, on le charge par `^Ldriver` (`<CONTROL>l driver`). Le fichier `driver.ll` doit se trouver dans le répertoire courant.

Exemple (avec Ingres) :

```
? (featurep 'driver)
= ()
? ^Ldriver
; Cloading ing_c.o
Chargement de DRIVER II
Chargement de DRIVER II Ok
DRIVER II v2.961105
Chargement des modeles a objets pour DRIVER
Chargement des modeles a objets pour DRIVER Ok
= driver.ll
? (featurep 'driver)
= driver
```

DriverVersion **[Variable]**

Contient le numéro de version de DRIVER. La partie entière est le numéro de version principal et la partie décimale est le numéro de sous-version (date).

Exemple:

```
? DriverVersion
= 2.961105
```

1 Exploitation de Driver comme serveur SGBD objet

1.1 Définition d'un SGBD

(DriverUseDBMSStratification
<ObjectDBMSName> <NF2DBMSName> <RelDBMSName>)
[SUBR à 3 arguments]

Définit la stratification de SGBD courante du serveur DRIVER. <ObjectDBMSName> est le nom du SGBD à objets à utiliser, <NF2DBMSName> le nom du SGBD NF2 sous-jacent et <RelDBMSName> le nom du SGBD relationnel sous-jacent au SGBD NF2 choisi. Chaque SGBD est retrouvé s'il existe déjà dans le serveur ou est créé dans le cas contraire. Toute base de données définie ultérieurement sera associée au SGBD courant de même modèle de donnée.

En règle générale, dans un serveur DRIVER ne sont définis qu'un seul SGBD à objets et qu'un seul SGBD NF2. Le SGBD relationnel pourra changer selon la localisation de la base de données relationnelles à utiliser.

Exemple:

```
? (DriverUseDBMSStratification 'ObjectDriver 'NF2Driver 'ingres)
= <ObjectDBMS ObjectDriver>
```

(DriverDefineDBMSDefaultObjectModel
<ObjectDBMSName> <ObjectModelName>)
[SUBR à 2 arguments]

Définit le modèle à objets de nom <ObjectModelName> comme modèle à utiliser par défaut dans le SGBD objet de nom <ObjectDBMSName>. Cette déclaration n'est valide que si le modèle est disponible dans le serveur (cf §4.1). Quand aucun modèle par défaut n'est défini à l'aide de cette fonction, c'est `MicroCeyx` qui est utilisé.

Exemple:

```
? (DriverDefineDBMSDefaultObjectModel 'ObjectDriver 'MicroCeyx)
= <ObjectModel MicroCeyx>
```

1.2 Gestion des utilisateurs

(DriverCryptPassword <PasswordString>) **[SUBR à 1 argument]**

Retourne la valeur cryptée du mot de passe <PasswordString>. Cette valeur cryptée est utilisable avec les fonctions de définition d'utilisateur (famille `DriverDefineUserStratification`).

Exemple:

```
? (DriverCryptPassword "APassword")
= 694
```

(DriverDefineUserStratification
<UserName> <CryptedPassword> <NF2UserName> <NF2CryptedPassword>
<RelUserName> <RelCryptedPassword>)

(DriverDefineObjectUserStratification
<UserName> <CryptedPassword> <NF2UserName> <NF2CryptedPassword>
<RelUserName> <RelCryptedPassword>)
[SUBR à 6 arguments]

Définit une stratification d'utilisateurs dans le serveur DRIVER. L'utilisateur de nom <UserName> et de mot de passe crypté <CryptedPassword> est retrouvé ou créé dans la strate objet, l'utilisateur de nom <NF2UserName> et de mot de passe crypté <NF2CryptedPassword> est retrouvé ou créé dans la strate NF2 et l'utilisateur de nom <RelUserName> et de mot de passe crypté <RelCryptedPassword> est retrouvé ou créé dans la strate relationnelle. Les trois utilisateurs sont ensuite associés et équivalents dans leurs strates respectives. La valeur de mot de passe crypté `()` définit l'utilisateur correspondant sans mot de passe.

Exemple:

```
? (DriverDefineObjectUserStratification
        'lebastard (DriverCryptPassword "MyPasswd")
        'lebastard (DriverCryptPassword "MyPasswd")
        'lebastar ())
= <ObjectUser lebastard>
```

(DriverDefineUserDefaultObjectModel <UserName> <ObjectModelName>)
[SUBR à 2 arguments]

Définit le modèle à objets de nom <ObjectModelName> comme modèle à utiliser par défaut pour l'utilisateur de nom <UserName> (strate objet). Cette déclaration n'est valide que si le modèle est disponible dans le serveur (cf §4.1).

Exemple:

```
? ^Ldriversmeci.lo
= driversmeci.lo
? (DriverDefineUserDefaultObjectModel 'lebastard 'SmeciObjectModel)
= <ObjectModel SmeciObjectModel>
```

1.3 Gestion des bases de données

(DriverDatabaseDefinitionMode
<ObjectDatabaseName> <NF2DatabaseName> <RelDatabaseName>)
[MACRO à 1, 2 ou 3 arguments]

(DriverObjectDatabaseDefinitionMode
<ObjectDatabaseName> <NF2DatabaseName> <RelDatabaseName>)
[SUBR à 3 arguments]

Définit la stratification de bases de données dans laquelle vont s'insérer les structures de données qui vont être décrites. <ObjectDatabaseName> est le nom de la base de données à objets à utiliser, <NF2DatabaseName> le nom de la base de données NF2 sous-jacente et <RelDatabaseName> le nom de la base de données relationnelle. Chaque base de données est retrouvée si elle existe déjà dans le SGBD courant correspondant ou est créée dans le cas contraire.

Exemple:

```
? ;;; équivaut à (DriverObjectDatabaseDefinitionMode 'demo 'demo 'demo)
? (DriverDatabaseDefinitionMode 'demo)
= t
```

(DriverCreateObjectPrimitiveType <TypeName> . <SuperTypeName>)
[SUBR à 1 ou 2 arguments]

Définissent dans la base de données NF2 courante un nouveau type primitif de nom <TypeName>. Si <SuperTypeName> est précisé, ce type est un sous-type du type <SuperTypeName>. Le nouveau type est également créé dans la base objet de la strate supérieure si elle existe. Cette fonction est strictement identique à la fonction `Driver-CreateNF2PrimitiveType`.

Exemple:

```
? (DriverCreateObjectPrimitiveType 'symbol 'string)
= <symbol Type>
```

Conversion de données

L'utilisateur peut définir les fonctions utilisées par le système pour transformer des données d'un type dans un autre. Ces fonctions ont pour nom `Driver_Type1_Type2`, prennent un argument qui est la donnée à convertir, de type `Type1`, et retourne la donnée convertie, de type `Type2`. L'absence de valeur est autorisée en entrée comme en sortie et est représentée par la valeur `()` (`nil`).

Exemple:

```
? (de Driver_string_symbol (String)
?     (if String (symbol () String)))
= Driver_string_symbol
? (de Driver_symbol_string (Symbol)
?     (if Symbol (string Symbol)))
= Driver_symbol_string
```

(DriverDefineClass <ClassName> . <Descriptions>) **[MACRO]**

Définit une classe de nom <ClassName> dans la base de données courante (sur le serveur DRIVER uniquement). La structure de données est créée dans la strate NF2 et, si elle existe, dans la strate objet : ainsi, **DriverDefineClass** et **DriverDefineNF2Collector** sont deux points d'entrée de la même fonction. Les <Descriptions> sont des listes à au moins deux éléments, le premier étant un mot-clé (symbol), les suivants définissant la valeur associée au mot-clé.

Un certain nombre de <Descriptions> sont ici possibles :

- `(SubclassOf SuperClassName)` précise que la classe en cours de description est une sous-classe de la classe de nom `SuperClassName`.
- La correspondance relationnelle de la classe (resp. du collecteur NF2) est principalement une table relationnelle logique (table élémentaire principale). Elle est précisée par une <Description> de la forme :

  `(MappedOn LogicalTableDescription)` ou
  `(MainlyMappedOn LogicalTableDescription)`

  où LogicalTableDescription peut être :
  - un nom de table relationnelle de la base relationnelle sous-jacente. La table logique s'identifie alors à cette table de même nom dans la base relationnelle.
  - une liste `(LogicalTableName TableName <Description>)`, laquelle définit une nouvelle table logique de nom `LogicalTableName`. `TableName` est le nom de la table de la base relationnelle sous-jacente sur laquelle est définie la table logique. <Description>, optionnelle, précise la table logique; la seule <Description> complémentaire possible ici est `(ReadOnly Boolean)` qui spécifie, quand `Boolean` vaut `t`, que la table logique est en lecture seulement. Par défaut, lecture et écriture sont autorisées.
  - un nom de table logique déjà définie.
- La <Description> du type de la classe (resp. du collecteur NF2) est obligatoire. Un type est une composition de constructeurs de type et de types. Certains termes sont nommés pour définir des slots. Les <Descriptions> suivantes sont disponibles :

  - `(TupleOf SlotName <Description1> ...<DescriptionN>)` définit un slot de nom `SlotName` et de type N-uplet. Les composants du n-uplet sont décrits par <Description1> ... <DescriptionN>. `SlotName` est facultatif et n'a de pas de

raison d'être utilisé si le n-uplet est le constructeur de plus haut niveau de la classe (resp. du collecteur NF2).
Les <Descriptions> optionnelles suivantes permettent de préciser la définition du slot :

- `(ReadOnly t)` indique que le slot (ici de type <`TupleType`>) est considéré par le SGBD comme n'étant accessible qu'en lecture pour l'utilisateur. En conséquence, toute éventuelle modification de son contenu sera ignoré lors de la validation de transaction.
- `(Volatile t)` indique que le slot n'est pas persistant. Quand la correspondance relationnelle de la classe est générée, aucune structure relationnelle ne lui est associée. Ce genre de slot est intéressant pour contenir les informations temporaires des objets (interface graphique, ...).

– `(TypeName SlotName . <Descriptions>)` définit un slot de nom `SlotName` et de type simple `TypeName`. `TypeName` peut être **Atomic**, **Reference**, **integer**, **float**, **string** ou tout autre type disponible dans le SGBD courant.
Reference permet de définir un slot de type "référence d'objet". Au niveau objet, le contenu d'un tel slot est un pointeur sur un objet. Au niveau NF2, il devient une clé étrangère de collecteur.

Les <Descriptions> optionnelles suivantes permettent de préciser la définition des slots :

- `(ReadOnly t)` indique que le slot est considéré par le SGBD comme n'étant accessible qu'en lecture pour l'utilisateur. En conséquence, toute éventuelle modification de son contenu sera ignoré lors de la validation de transaction.
- `(Volatile t)` indique que le slot n'est pas persistant. Quand la correspondance relationnelle de la classe est générée, aucune structure relationnelle ne lui est associée. Ce genre de slot est intéressant pour contenir les informations temporaires des objets (interface graphique, ...).
- `(MappedOn LogicalAttributeDescription)` permet de préciser la correspondance relationnelle du slot, à savoir un attribut relationnel. `LogicalAttributeDescription` est de la forme :

  ```
  (LogicalTableDescription AttributeName)
  ```

  où `LogicalTableDescription` est une description de table logique comme précédemment définie et `AttributeName` est un nom d'attribut de cette table.
- `(RelConstraint Lambda LogicalAttributeDescription)` permet de poser une contrainte sur la correspondance relationnelle du slot. La `Lambda`, à un seul argument, est l'expression de la contrainte : si l'application de

la lambda à une valeur de l'attribut retourne un résultat différent de `()`, la contrainte est respectée; dans le cas contraire, elle est violée. `Logical-AttributeDescription` est facultative.

- `(ReferTo ClassName)` permet de préciser la classe (resp. le collecteur NF2) des objets (resp. des données complexes) référencé(e)s quand le slot est de type "référence d'objet".
- `(Inverse (ClassName Slot1Name ... SlotNName))` permet de préciser, quand un slot est de type "référence d'objet", l'éventuel slot `SlotNName` de la classe référencée qui est le lien inverse du slot en cours de définition. `(ClassName Slot1Name ... SlotNName)` doit se lire comme l'accès absolu du slot inverse `SlotNName` dans sa classe, soit `ClassName.Slot1Name.(...).SlotNName`.

Remarques :

* Le mot-clé `Atomic` n'est autorisé comme type de slot que si la <Description> `MappedOn` est précisée. Le type du slot est alors le type de l'attribut associé.
* Si le type du slot et le type de l'attribut relationnel sont différents, les convertisseurs (cf. page 9) de données définis à cet effet seront utilisés pour convertir les données d'un type dans l'autre lors de l'exploitation.

– `(CollectionTypeName SlotName . <Descriptions>)` où `CollectionTypeName` est un mot-clé parmi `BagOf`, `SetOf`, `ListOf` et `ArrayOf` définit un slot de nom `SlotName` et de type collection (plus précisément de type bag, set, list ou array). Chaque collection est homogène, tous ses éléments sont du même type. L'élément de collection doit obligatoirement être décrit par l'une des <`Descriptions`>; les autres permettent de préciser la définition du slot :

- `(ReadOnly t)` indique que le slot est considéré par le SGBD comme n'étant accessible qu'en lecture pour l'utilisateur. En conséquence, toute éventuelle modification de son contenu sera ignoré lors de la validation de transaction.
- `(Volatile t)` indique que le slot n'est pas persistant. Quand la correspondance relationnelle de la classe est générée, aucune table relationnelle ne lui est associée. Ce genre de slot est intéressant pour contenir les informations temporaires des objets (interface graphique, ...).
- `(MappedOn LogicalTableDescription)` permet de préciser sa correspondance relationnelle, à savoir une table logique. Cette table logique, support des collections, doit obligatoirement être rattachée à la collection propriétaire ou à la classe (resp. le collecteur NF2) par une jointure qui doit être décrite dans une autre <Descriptions> complémentaire de la même collection.

- `(RelConstraint Lambda LogicalAttribute1Description ... LogicalAttributeNDescription)` permet de poser une contrainte sur la correspondance relationnelle de la collection, à savoir la table logique associée. La `Lambda`, à N arguments, est l'expression de la contrainte : si l'application de la lambda aux valeurs des attributs `LogicalAttribute1Description ...LogicalAttributeNDescription` retourne un résultat différent de `()`, la contrainte est respectée; dans le cas contraire, elle est violée.

- `(JoinDescription LogicalTable1Description LogicalTable2Description Lambda LogicalAttribute1Description ... LogicalAttributeNDescription)` permet de définir une jointure entre les tables logiques `LogicalTable1Description` et `LogicalTable2Description`. La `Lambda`, à N arguments, définit l'expression de la jointure et ses arguments représentent les N attributs `LogicalAttributePDescription`.
  Remarques :

  * Si `LogicalTable2Description` doit être la correspondance principale de la collection en cours de description et si `LogicalTable1Description` est l'une des tables élémentaires de la collection propriétaire (éventuellement la classe, resp. le collecteur NF2), cette jointure précise comment atteindre les éléments de la collection courante à partir des éléments de la collection propriétaire.

  * Si `LogicalTable1Description` est une des tables élémentaires de la collection en cours de description et si `LogicalTable2Description` doit être une nouvelle table élémentaire de cette même collection en cours de description, la jointure précise comment associer un n-uplet de `LogicalTable1Description` à un n-uplet de `LogicalTable2Description` pour chaque élément de la collection courante.

- `(OrderedBy (LogicalAttribute1Description Order1) ... (LogicalAttributeNDescription OrderN)` permet de définir des critères de tri primaire, secondaire, etc. Pour chacun, le tri peut être ascendant ou descendant (`OrderP` vaut `Ascending` ou `Descending`). Cette <Description> n'a pas de sens pour définir la correspondance d'une collection de type set. Elle est par contre obligatoire pour définir la correspondance des collections de type bag, list et array.

– `(Computation SlotName . <Descriptions>)` définit un slot de nom `SlotName` et de type calcul.

Les <Descriptions> permettent de préciser la définition du slot :

- `(ComputationType CType)` permet de préciser la nature du calcul, à savoir mono–n-uplet ou multi–n-uplets (`CType` vaut `MonoTupleComputation` ou `MultiTupleComputation`).
- `(DefinedBy Lambda LogicalAttribute1Description ... LogicalAttributeNDescription)` définit le calcul. La `Lambda`, à N arguments, définit son expression et ses arguments représentent les N attributs `LogicalAttributePDescription`. Une expression de calcul mono–n-uplet peut utiliser les quatre opérations $+$, $-$, $*$ et $/$. Une expression de calcul multi-n-uplets peut en plus faire intervenir les fonctions `average` (moyenne) et `sum` (somme).

- La correspondance relationnelle de la classe (resp. du collecteur NF2) peut, outre la table élémentaire principale, comprendre d'autres tables élémentaires, dites tables élémentaires secondaires. Des jointures doivent alors permettre d'atteindre depuis la table principale, de proche en proche, toutes les tables élémentaires. Chaque jointure doit être décrite par une <Descriptions> :

```
(JoinDescription
     LogicalTable1Description LogicalTable2Description
     Lambda
     LogicalAttribute1Description ... LogicalAttributeNDescription)
```

  qui définit une jointure entre la table `LogicalTable1Description` et la table `LogicalTable2Description`. La `Lambda`, à N arguments, définit l'expression de la jointure et chacun de ses arguments représente les N attributs `LogicalAttributePDescription`.

Exemples de définition :

```
(DriverDefineClass employee
   (TupleOf
      (symbol name)
      (symbol firstname)
      (symbol jobdesc)
      (Reference manager
         (ReferTo employee))
      (float salary
         (RelConstraint
             (lambda (s) (and (>= s 700.) (<= s 9999.)))))
```

```
      (Reference dept
         (ReferTo department)
         (Inverse (department employees)))))

; autre description possible de la même classe
; avec définition du mapping
(DriverDefineClass employee
   (MainlyMappedOn emp)
   (TupleOf
       (symbol name
           (MappedOn (emp ename)))
       (symbol firstname
           (MappedOn (emp fname)))
       (symbol jobdesc
           (MappedOn (emp job)))
       (Atomic manager
           (ReferTo employee)
           (MappedOn (emp mgr)))
       (Atomic salary
           (MappedOn (emp sal))
           (RelConstraint
               (lambda (s) (and (>= s 700.) (<= s 9999.)))))
       (Atomic dept
           (MappedOn (emp deptn))
           (ReferTo department)
           (Inverse (department employees)))))

(DriverDefineClass salesman
   (SubclassOf employee)
   (TupleOf
       (symbol jobdesc
           (RelConstraint
               (lambda (a) (eq a 'salesman))))
       (Atomic commission
           (MappedOn (emp com)))
       (Computation income
           (DefinedBy (lambda (a b) (+ a b))
                      (emp sal) (emp com)))))
```

```
(DriverDefineClass department
   (MainlyMappedOn dept)
   (TupleOf
       (symbol name
           (MappedOn (dept dname)))
       (SetOf employees
           (MappedOn (e emp))
           (Atomic
               (MappedOn (e empno))
               (ReferTo employee))
           (ReadOnly t)
           (Inverse (employee dept))
           (JoinDescription dept e
                (lambda (a1 a2) (eq a1 a2))
                (dept deptno) (e deptn)))
       (ListOf sites
           (symbol sname
               (MappedOn (deptsite sname)))
           (MappedOn deptsite)
           (OrderedBy ((deptsite sname) Ascending)))
       (Computation salaryaverage
           (ComputationType MultiTupleComputation)
           (DefinedBy (lambda (a) (average a)) (emp sal))))
   (JoinDescription dept emp
        (lambda (a1 a2) (eq a1 a2))
        (dept deptno) (emp deptn))
   (JoinDescription dept deptsite
        (lambda (a1 a2) (eq a1 a2))
        (dept deptno) (deptsite dept)))
```

(DriverMakeAllClassesPersistent) **[SUBR sans argument]**

Génère les tables relationnelles et les correspondances nécessaires à la persistance dans la base de données relationnelle des objets des classes de la base de données à objets courante. Si certaines classes ou certaines parties de classes sont déjà associées à des structures relationnelles, ces correspondances sont conservées et complétées. Les slots définis par l'utilisateur comme volatiles sont ignorés par le générateur.

La fonction **DriverNewRelationalDatabaseDefinition** peut être avantageusement utilisée conjointement avec **DriverMakeAllClassesPersistent** afin de générer toujours les mêmes correspondances relationnelles pour un ensemble de classes donné.

Exemple:

```
? (DriverMakeAllClassesPersistent)
= t
```

(DriverMakeClassPersistent <ClassName>) **[SUBR à 1 argument]**

Génère les tables relationnelles et les correspondances nécessaires à la persistance dans la base de données relationnelle des objets de la classe de nom <ClassName> de la base de données à objets courante. Si certaines parties de la classe sont déjà associées à des structures relationnelles, ces correspondances sont conservées et complétées. Les slots définis par l'utilisateur comme volatiles sont ignorés par le générateur.

La fonction **DriverNewRelationalDatabaseDefinition** peut être avantageusement utilisée conjointement avec **DriverMakeClassPersistent** afin de générer toujours les mêmes correspondances relationnelles pour une classe donnée.

Exemple:

```
? (DriverMakeClassPersistent 'employee)
= t
```

(DriverDatabaseDefinitionModeEnd) **[SUBR sans argument]**
(DriverObjectDatabaseDefinitionModeEnd) **[SUBR sans argument]**

Clôt le mode de définition de base de données.

Exemple:

```
? (DriverDatabaseDefinitionModeEnd)
= t
```

1.4 Exploitation des données

1.4.1 Authentification de l'utilisateur

L'exploitation du serveur DRIVER n'est possible qu'après authentification de l'utilisateur.

(DriverConnectLocalUser <UserName> <Password>)
(DriverConnectObjectLocalUser <ObjectUserName> <Password>)
[SUBR à 2 arguments]

Authentifie l'utilisateur <UserName> (strate objet) par son mot de passe <Password>.

Exemple:

```
? (DriverConnectLocalUser 'lebastard "MyPasswd")
= t
```

1.4.2 Gestion des transactions

Un utilisateur authentifié ne peut accéder aux données qu'à l'intérieur d'une transaction.

(DriverCreateTransaction <ObjectDatabaseName>)
(DriverCreateObjectTransaction <ObjectDatabaseName>)
[SUBR à 1 argument]

Crée et retourne une transaction objet sur la base de données de nom <ObjectDatabaseName>. Les opérations sur cette transaction seront déclenchées par envois de message à l'objet transaction.

Exemple:

```
? (setq Tr (DriverCreateTransaction 'demo))
= <ObjectTransaction <ObjectUser lebastard>, <ObjectDatabase demo»
```

(send 'Begin <Transaction>) **[MESSAGE]**

Ouvre la transaction <Transaction>. Après ouverture de la transaction, l'accès aux données peut commencer. La transaction ouverte devient la transaction courante.

Exemple:

```
? (send 'Begin Tr)
= t
```

(DriverCurrentTransaction)
(DriverCurrentObjectTransaction) **[SUBR sans argument]**

Retourne la transaction objet courante, c'est-à-dire la dernière ouverte (et non la dernière créée).

Exemple:

```
? (DriverCurrentTransaction)
= <ObjectTransaction <ObjectUser lebastard>, <ObjectDatabase demo»
```

(send 'Commit <Transaction>) **[MESSAGE]**

Valide la transaction <Transaction>. La validation par Commit se termine par le relâchement de tous les verrous et par la destruction de l'objet <Transaction>.

Exemple:

```
? (send 'Commit Tr)
= t
```

(send 'Rollback <Transaction>) **[MESSAGE]**

Annule la transaction <Transaction>. L'annulation se termine par le relâchement de tous les verrous et par la destruction de l'objet <Transaction>.

Exemple:

```
? (send 'Rollback Tr)
= t
```

1.4.3 L'accès aux données

L'accès aux objets, aux données les constituant et l'attribution de la persistance se font par les fonctions suivantes :

(DriverQuery <OQLQuery>) **[SUBR à 1 argument]**

Permet d'accéder à la base de données objet de la transaction courante à l'aide du langage OQL défini par la norme ODMG. <OQLQuery> doit être une chaîne de caractères constituant une requête OQL.

Exemple:

```
? (DriverQuery "select e from e in employee;")
= (<object martin> <object miller> <object james> <object ward> <object allen>
 <object jones> <object blake> <object clark> <object ford> <object scott>
 <object king>)
? (DriverQuery "select e.name from e in employee where e.salary>1200. and
 e.salary<2000.;")
= (ward martin james)
? (DriverQuery "select struct(n: e.name, f: e.firstname, p: e.phone) from e
 in employee where e.salary > 1000. and e.salary < 2000.;")
= (#[martin georges 40-276951] #[allen jack 78-383726] #[miller paul 43-127772]
 #[james peter 40-449323] #[ward peter 40-378467])
? (DriverQuery "select struct(n: e.name, f: e.firstname, p: e.phone) from e
 in salesman where e.salary > 1000. and e.salary < 2000.;")
= (#[martin georges 40-276951] #[allen jack 78-383726] #[james peter 40-449323]
 #[ward peter 40-378467])
? (DriverQuery "select struct(emp: e.name, man: e.manager.name) from e in
 employee;")
= (#[ford jones] #[scott jones] #[allen blake] #[james blake] #[martin blake]
 #[ward blake] #[miller clark] #[blake king] #[clark king] #[jones king]
 #[smith ford])
? (DriverQuery "select struct(emp: e, man: e.manager) from e in employee;")
= (#[<object ford> <object jones>] #[<object scott> <object jones>] #[<object
 allen> <object blake>] #[<object james> <object blake>] #[<object martin>
 <object blake>] #[<object ward> <object blake>] #[<object miller> <object
 clark>] #[<object blake> <object king>] #[<object clark> <object king>]
 #[<object jones> <object king>] #[<object smith> <object ford>])
? (DriverQuery "select e from e in employee where e.dept.name=""sales"";")
= (<object martin> <object blake> <object allen> <object james> <object ward>)
? (DriverQuery "select e from e in employee where e.manager.name=""blake"";")
= (<object martin> <object allen> <object james> <object ward>)
```

(DriverLoadObject <UserObject>) **[SUBR à 1 argument]**

(Re)Charge en mémoire l'objet <UserObject> à l'aide des données contenues dans le SGBD sous-jacent. Si <UserObject> est encore sous la forme d'un défaut d'objet, remplace le défaut par l'objet qu'il représentait.

Exemple:

```
? (setq d (DriverQuery "select distinct e.dept from e in employee;"))
= (<An object default> <An object default> <An object default>)
? (DriverLoadObject (car d))
= <object accounting>
? d
= (<object accounting> <An object default> <An object default>)
```

(DriverMakeObjectPersistent <UserObject>) **[SUBR à 1 argument]**

Rend persistant l'objet <UserObject>. Tous les objets référencés directement ou indirectement par des slots persistants de <UserObject> deviennent également persistants s'ils ne l'étaient pas déjà.

Exemple:

```
? (DriverMakeObjectPersistent
?    (omakeq {department} name 'factory))
= <object factory>
```

(DriverRemoveObjectPersistency <UserObject>) **[SUBR à 1 argument]**

Rend volatile l'objet <UserObject>. L'objet et toutes ses références sont supprimés de la base de données à la validation de la transaction.

Exemple:

```
? (DriverRemoveObjectPersistency
```

```
?    (car (DriverQuery
?      "select e from e in employee where e.name=""Miru"";")))
= <object Miru>
```

(DriverCreateClass <ClassName>) **[SUBR à 1 argument]**

Crée dynamiquement la classe <ClassName> dans le modèle à objets de la transaction courante. Cette fonction est utile pour exécuter des requêtes OQL qui doivent retourner des objets dont la classe n'a pas été créée par l'application.

Exemple:

```
? (DriverCreateClass 'project)
= t
? (omakeq {project} name 'alpha)
= <object alpha>
```

(DriverCreateClasses) **[SUBR sans argument]**

Crée dynamiquement dans le modèle à objets de la transaction courante toutes les classes de la base à objets accédée dans cette même transaction. Comme la précédente, cette fonction est utile pour exécuter des requêtes OQL qui doivent retourner des objets dont les classes n'ont pas été créées par l'application.

Exemple:

```
? (DriverCreateClasses)
= t
```

2 Exploitation de Driver comme serveur SGBD NF2

2.1 Définition d'un SGBD

(DriverUseNF2DBMSStratification <NF2DBMSName> <RelDBMSName>) **[SUBR à 2 arguments]**

Définit la stratification de SGBD courante du serveur DRIVER quand la strate objet n'est pas utilisée. <NF2DBMSName> est le nom du SGBD NF2 à utiliser et <RelDBMSName> le nom du SGBD relationnel sous-jacent. Chaque SGBD est retrouvé s'il existe déjà dans le serveur ou est créé dans le cas contraire.

Exemple:

```
? (DriverUseNF2DBMSStratification 'NF2Driver 'ingres)
= <NF2DBMS NF2Driver>
```

2.2 Gestion des utilisateurs

(DriverDefineNF2UserStratification
<NF2UserName> <NF2CryptedPassword> <RelUserName>
<RelCryptedPassword>)
[SUBR à 4 arguments]

Définit une stratification d'utilisateurs dans le serveur DRIVER. L'utilisateur de nom <NF2UserName> et de mot de passe crypté <NF2CryptedPassword> est retrouvé ou créé dans la strate NF2 et l'utilisateur de nom <RelUserName> et de mot de passe crypté <RelCryptedPassword> est retrouvé ou créé dans la strate relationnelle. Les deux utilisateurs sont ensuite associés et équivalents dans leurs strates respectives. La valeur de mot de passe crypté `()` définit l'utilisateur correspondant sans mot de passe.

Exemple:

```
? (DriverDefineNF2UserStratification
        'lebastard (DriverCryptPassword "MyPasswd")
        'lebastar ())
= <NF2User lebastard>
```

2.3 Gestion des bases de données

(DriverNF2DatabaseDefinitionMode
<NF2DatabaseName> <RelDatabaseName>)
[SUBR à 2 arguments]

Définit la stratification de bases de données dans laquelle vont s'insérer les structures de données qui vont être décrites quand la strate objet n'est pas utilisée. <NF2DatabaseName> est le nom de la base de données NF2 et <RelDatabaseName> le nom de la base de données relationnelle sous-jacente. Chaque base de données est retrouvée si elle existe déjà dans le SGBD courant correspondant ou est créée dans le cas contraire.

Exemple:

```
? (DriverNF2DatabaseDefinitionMode 'demo 'ingresdemo)
= t
```

(DriverCreateNF2PrimitiveType <TypeName> . <SuperTypeName>)
[SUBR à 1 ou 2 arguments]

Définissent dans la base de données NF2 courante un nouveau type primitif de nom <TypeName>. Si <SuperTypeName> est précisé, ce type est un sous-type du type <SuperTypeName>. Le nouveau type est également créé dans la base objet de la strate supérieure si elle existe. Cette fonction est strictement identique à la fonction `DriverCreateObjectPrimitiveType`. Les convertisseurs de type doivent être définis comme il est spécifié page 9.

Exemple:

```
? (DriverCreateNF2PrimitiveType 'lisp 'string)
= <lisp Type>
? (de Driver_string_lisp (String)
?     (if String (read-from-string String)))
= Driver_string_lisp
? (de Driver_lisp_string (Lisp)
?     (if Lisp (print-to-string Lisp)))
= Driver_lisp_string
```

(DriverDefineNF2Collector <NF2CollectorName> . <Descriptions>) **[MACRO]**

Définit un collecteur NF2 de nom <NF2CollectorName> dans la base de données courante (sur le serveur DRIVER uniquement). La structure de données est créée dans la strate NF2

et, si elle existe, dans la strate objet : ainsi, `DriverDefineClass` et `DriverDefineNF2-Collector` sont deux points d'entrée de la même fonction.

Voir la description de `DriverDefineClass` pour davantage d'informations (remplacer "classe" par "collecteur NF2").

(DriverMakeAllNF2CollectorsPersistent) **[SUBR sans argument]**

Génère les tables relationnelles et les correspondances nécessaires à la persistance dans la base de données relationnelle des données des collecteurs NF2 de la base de données NF2 courante. Si certains collecteurs ou certaines parties de collecteurs sont déjà associées à des structures relationnelles, ces correspondances sont conservées et complétées. Les slots NF2 définis par l'utilisateur comme volatiles sont ignorés par le générateur.

La fonction **DriverNewRelationalDatabaseDefinition** peut être avantageusement utilisée conjointement avec **DriverMakeAllNF2CollectorsPersistent** afin de générer toujours les mêmes correspondances relationnelles pour un ensemble de collecteurs donné.

Exemple:

```
? (DriverMakeAllNF2CollectorsPersistent)
= t
```

(DriverMakeNF2CollectorPersistent <NF2CollectorName>)
[SUBR à 1 argument]

Génère les tables relationnelles et les correspondances nécessaires à la persistance dans la base de données relationnelle des données du collecteur NF2 de nom <NF2CollectorName> de la base de données NF2 courante. Si certaines parties du collecteur sont déjà associées à des structures relationnelles, ces correspondances sont conservées et complétées. Les slots NF2 définis par l'utilisateur comme volatiles sont ignorés par le générateur.

La fonction **DriverNewRelationalDatabaseDefinition** peut être avantageusement utilisée conjointement avec **DriverMakeNF2CollectorPersistent** afin de générer toujours les mêmes correspondances relationnelles pour un collecteur donné.

Exemple:

```
? (DriverMakeNF2CollectorPersistent 'employee)
= t
```

(DriverNF2DatabaseDefinitionModeEnd) **[SUBR sans argument]**

Clôt le mode de définition de base de données.

Exemple:

```
? (DriverNF2DatabaseDefinitionModeEnd)
= t
```

2.4 Exploitation des données

2.4.1 Authentification de l'utilisateur

(DriverConnectNF2LocalUser <NF2UserName> <Password>)
[SUBR à 2 arguments]

Authentifie l'utilisateur NF2 <NF2UserName> par son mot de passe <Password>.

Exemple:

```
? (DriverConnectNF2LocalUser 'lebastard "MyPasswd")
= t
```

2.4.2 Gestion des transactions

(DriverCreateNF2Transaction <NF2DatabaseName>) **[SUBR à 1 argument]**

Crée et retourne une transaction NF2 sur la base de données de nom <NF2Database-Name>. Les opérations sur cette transaction seront déclenchées par envois de message à l'objet transaction.

Exemple:

```
? (DriverCreateNF2Transaction 'demo)
= <NF2Transaction <NF2User lebastard>, <NF2Database demo»
```

(send 'Begin <NF2Transaction>) **[MESSAGE]**
(send 'Commit <NF2Transaction>) **[MESSAGE]**
(send 'Rollback <NF2Transaction>) **[MESSAGE]**

Tous les messages de transaction sont utilisables avec les transactions NF2.

(DriverCurrentNF2Transaction) **[SUBR sans argument]**

Retourne la transaction NF2 courante, c'est-à-dire la dernière ouverte (et non la dernière créée).

Exemple:

```
? (DriverCurrentNF2Transaction)
= <NF2Transaction <NF2User lebastard>, <NF2Database demo»
```

2.4.3 L'accès aux données

(DriverNF2Query <OQLQuery>) **[SUBR à 1 argument]**

Permet d'accéder à la base de données NF2 de la transaction courante à l'aide du langage OQL défini par la norme ODMG. <OQLQuery> doit être une chaîne de caractères constituant une requête OQL.

Exemple:

```
? (DriverNF2Query "select e from e in employee;")
= (#[7654 martin georges 234FS1 salesman 7698 1250. 2 30 40-276951 3005] #[7934
 miller paul 95Z0H5 clerk 7782 1300. 2 10 43-127772 1276] #[7655 james peter
 K4G262 salesman 7698 1350. 2 30 40-449323 ()] #[7521 ward peter 128347 salesman
 7698 1250. 2 30 40-378467 1253] #[7499 allen jack 76373Y salesman 7698 1600. 3
 30 78-383726 2004] #[7566 jones eric B7C123 manager 7839 2975. 4 20 40-783539
 1975] #[7698 blake harold A473SE manager 7839 2850. 4 30 44-183211 1429] #[7782
 clark john 462SQ1 manager 7839 2500. 4 10 40-893723 1230] #[7902 ford john
 6D3210 analyst 7566 3000. 4 20 40-374845 4328] #[7788 scott pit XA435C analyst
 7566 3000. 4 20 44-003231 1928] #[7839 king paul A34F4 president () 5000. 5 10
 40-223233 1230])
```

```
? (DriverNF2Query "select e.name from e in employee where e.salary>1200. and
 e.salary<2000.;")
= (ward martin james)
? (DriverNF2Query "select struct(n: e.name, f: e.firstname, p: e.phone) from e
 in employee where e.salary > 1000. and e.salary < 2000.;")
= (#[martin georges 40-276951] #[allen jack 78-383726] #[miller paul 43-127772]
 #[james peter 40-449323] #[ward peter 40-378467])
```

3 Exploitation de Driver comme serveur SGBD relationnel

3.1 Gestion des bases de données

(DriverRelationalDatabaseDefinitionMode <RelDatabaseName>)
[SUBR à 1 argument]

Définit la base de données relationnelle dans laquelle vont s'insérer les structures de données qui vont être décrites quand les strates objet et NF2 ne sont pas utilisées. La base de données est retrouvée si elle existe déjà dans le SGBD courant correspondant ou est créée dans le cas contraire.

Exemple:

```
? (DriverRelationalDatabaseDefinitionMode 'ingresdemo)
= t
```

(DriverCreateRelationalType <TypeName> . <SuperTypeName>)
[SUBR à 1 ou 2 arguments]

Définit dans la base de données relationnelle courante un nouveau type de nom <TypeName>. Si <SuperTypeName> est précisé, ce type est un sous-type du type <SuperTypeName>. Le nouveau type est également créé dans les bases des strates supérieures (NF2 et objet) si elles existent.

Exemple:

```
? (DriverCreateRelationalType 'money)
= <money Type>
? (DriverCreateRelationalType 'float4 'float)
= <float4 Type>
```

(DriverNewRelationalDatabaseDefinition) **[SUBR sans argument]**

Supprime de la base de données relationnelle courante toutes les définitions de table existantes (uniquement dans le serveur DRIVER). Cette fonction est utile pour nettoyer une base de données relationnelle avant génération de correspondances de classes ou de collecteurs NF2. Ce nettoyage permet en particulier de générer les mêmes correspondances lors de tout chargement d'un schéma objet ou NF2 donné.

Exemple:

```
? (DriverNewRelationalDatabaseDefinition)
= t
```

(DriverDefineRelationalTable <TableName> <Attribute1> ... <AttributeN>) **[MACRO]**

Définit la table relationnelle de nom <TableName> dans la base de données relationnelle courante (dans le serveur DRIVER uniquement).

<Attribute1> ... <AttributeN> décrivent les attributs de la table. Ces descriptions sont des listes à quatre éléments :

(<AttributeName> <AttributeType> <TypeLength> <Status>)

<AttributeType> peut être tout type existant de la base de données relationnelle courante. Par défaut, les types existant sont `integer`, `float` et `string`.

<Status> peut être un statut parmi `()` (status commun), `KeyPart`, `NotNull` et `Unique`.

Exemple de définition :

```
(DriverDefineRelationalTable emp
   (empno integer 2 KeyPart)
   (ename string 20 ())
   (fname string 20 ())
   (job   string 20 ())
   (mgr   integer 2 ())
   (sal   float   8 ())
   (com   float   8 ())
   (deptn integer 2 ()))
```

(DriverRelationalDatabaseDefinitionModeEnd) **[SUBR sans argument]**

Clôt le mode de définition de base de données.

Exemple:

```
? (DriverRelationalDatabaseDefinitionModeEnd)
= t
```

3.2 Exploitation des données

3.2.1 Authentification de l'utilisateur

(DriverConnectRelationalLocalUser <RelUserName> <Password>)
[SUBR à 2 arguments]

Authentifie l'utilisateur relationnel <RelUserName> par son mot de passe <Password>.

Exemple:

```
? (DriverConnectRelationalLocalUser 'lebastar ())
= t
```

3.2.2 Gestion des transactions

(DriverCreateRelationalTransaction <RelDatabaseName>) **[SUBR à 1 argument]**

Crée et retourne une transaction relationnelle sur la base de données de nom <RelDatabaseName>. Les opérations sur cette transaction seront déclenchées par envois de message à l'objet transaction.

Exemple:

```
? (DriverCreateRelationalTransaction 'demo)
= <RelTransaction <RelUser lebastard>, <RelDatabase demo»
```

(DriverCurrentRelationalTransaction) **[SUBR sans argument]**

Retourne la transaction relationnelle courante, c'est-à-dire la dernière ouverte (et non la dernière créée).

Exemple:

```
? (DriverCurrentRelationalTransaction)
= <RelTransaction <RelUser lebastard>, <RelDatabase demo»
```

4 Utilisation avancée

4.1 Gestion des modèles à objets

Par défaut, DRIVER fonctionne avec le modèle à objets **MicroCeyx**. Cependant, il est possible de prendre en compte d'autres modèles.

Un modèle à objets est disponible sur le serveur DRIVER si la classe système le représentant a été définie. Cette définition comprend un certain nombre de méthodes qui doivent impérativement être décrites avant toute utilisation du modèle. Ces méthodes sont présentées ci-après.

4.1.1 Définition d'un nouveau modèle

On introduit un nouveau modèle à objets de nom "`ObjectModelName`" dans le serveur DRIVER en définissant une nouvelle sous-classe "`ObjectModelName`" à la classe système **DriverObjectModel**.

On effectue cette définition par la commande :

```
(deftclass {DriverObjectModel}:ObjectModelName)
```

Définition du modèle "MicroCeyx" :

```
(deftclass {DriverObjectModel}:MicroCeyx)
```

4.2 Définition des méthodes

Les méthodes suivantes doivent impérativement être décrites pour toute utilisation du modèle :

({ObjectModelName}:ObjectMake <ObjectModelName> <ObjectDBMS>) **[SUBR à 2 arguments]**

Pseudo-méthode appelée pour créer une instance de modèle à objets de classe `Object-ModelName`. <ObjectModelName> est le nom du modèle à objets et <ObjectDBMS> est le SGBD objet pour lequel est créé le modèle.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:ObjectMake (ObjectModelName ObjectDBMS)
    ({DriverObjectModel}:Init
        (omakeq {MicroCeyx}
                ObjectDBMS ObjectDBMS
                Name ObjectModelName)))
```

({ObjectModelName}:Delete <ObjectModel>) **[SUBR à 1 argument]**

Méthode appelée lors de la destruction du modèle à objets <ObjectModel>.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:Delete (MicroCeyx)
    ({DriverObjectModel}:Delete MicroCeyx))
```

({ObjectModelName}:MakeObject <ObjectModel> <Class>) **[SUBR à 2 arguments]**

Méthode appelée pour créer dans le modèle à objets un objet utilisateur instance de la classe représentée en DRIVER par l'objet système <Class>.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:MakeObject (ObjectModel Class)
    (funcall (getfn (get-abbrev ({DriverType}:GetName Class))
             'make)))
```

({ObjectModelName}:ClassInstanceP <ObjectModel> <Object> <Class>)
[SUBR à 3 arguments]

Méthode appelée pour tester si l'objet utilisateur <Object> est instance de la classe représentée en DRIVER par l'objet système <Class>.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:ClassInstanceP (ObjectModel Object Class)
    (subtypep (type-of Object)
              (get-abbrev ({DriverType}:GetName Class))))
```

({ObjectModelName}:ChangeObjectClass
<ObjectModel> <NewClass> <Object>)
[SUBR à 3 arguments]

Méthode appelée pour transformer un objet utilisateur <Object> en instance de la classe représentée en DRIVER par l'objet système <NewClass>.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:ChangeObjectClass (ObjectModel NewClass Object)
    (let ((NewObject ({MicroCeyx}:MakeObject ObjectModel NewClass)))
         (if (or (typep NewObject (type-of Object))
                 (typep Object (type-of NewObject)))
             (:CopyInto Object NewObject))
         (exchvector Object NewObject)))

(de :CopyInto (Object1 Object2)
    (let ((i -1))
         (repeat (imin (vlength Object1) (vlength Object2))
                 (vset Object2 (incr i) (vref Object1 i)))
         Object2))
```

({ObjectModelName}:BuildObject <ObjectModel>
<Class> <UserObject> <NF2Type> <NF2TypeData> <ObjectTransaction>)
[SUBR à 6 arguments]

Méthode appelée pour construire un objet utilisateur <UserObject> de classe <Class> à partir de la donnée NF2 <NF2TypeData> de type <NF2Type>.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:BuildObject
       (ObjectModel Class UserObject NF2Type NF2TypeData ObjectTransaction)
    (ifn ({DriverType}:ObjectTypeP ({Class}:GetType Class))
         (:DumpObject UserObject NF2Type NF2TypeData)
         (send 'BuildObjectTypeData
               ({DriverSlot}:GetType
                   ({CollectionType}:GetSlot
                       ({Class}:GetType Class)))
               UserObject NF2Type NF2TypeData ObjectTransaction)))

(de :DumpObject (UserObject NF2Type NF2TypeData)
    (let ((NewObjectBody
             ({DriverType}:CopyData NF2Type NF2TypeData)))
         (typevector NewObjectBody (type-of UserObject))
         (exchvector UserObject NewObjectBody)))
```

({ObjectModelName}:BuildNF2Data <ObjectModel> <Class> <NF2Type> <UserObject> <ObjectTransaction>) [SUBR à 5 arguments]

Méthode appelée pour construire la donnée NF2 de type <NF2Type> correspondant à un objet utilisateur <UserObject> de classe <Class>.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:BuildNF2Data
       (ObjectModel Class NF2Type UserObject ObjectTransaction)
    (ifn ({DriverType}:ObjectTypeP ({Class}:GetType Class))
         ({DriverType}:CopyData
             ({DriverSlot}:GetType
                 ({CollectionType}:GetSlot
                     ({Class}:GetType Class)))
```

```
            UserObject)
        (send 'BuildNF2TypeData
              ({DriverSlot}:GetType
                  ({CollectionType}:GetSlot
                      ({Class}:GetType Class)))
              NF2Type UserObject ObjectTransaction)))
```

({ObjectModelName}:MakeClass <ObjectModel> <Class>)

[SUBR à 2 arguments]

Méthode appelée pour créer dans le modèle à objets la classe représentée en DRIVER par l'objet système <Class>.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:MakeClass (ObjectModel Class)
    (let ((SuperClassSlots
            (if ({DriverType}:GetSuper Class)
                ({TupleType}:GetSlots
                    (:MustBeAMicroCeyxTupleType
                        ({DriverSlot}:GetType
                            ({CollectionType}:GetSlot
                                ({Class}:GetType
                                    ({DriverType}:GetSuper Class)))))))))
          (apply 'deftclass
                 (cons (:GetMicroCeyxType Class)
                       (mapcan (lambda (Slot)
                                 (ifn (:SlotMemq ({DriverSlot}:GetName Slot)
                                                 SuperClassSlots)
                                      (list ({DriverSlot}:GetName Slot))))
                               ({TupleType}:GetLocalSlots
                                   (:MustBeAMicroCeyxTupleType
                                       ({DriverSlot}:GetType
                                           ({CollectionType}:GetSlot
                                               ({Class}:GetType Class))))))))))

(de :SlotMemq (Name Slots)
    (if Slots
        (if (eq ({DriverSlot}:GetName (car Slots)) Name)
           Slots (:SlotMemq Name (cdr Slots)))))
```

```
(de :GetMicroCeyxType (Class)
    (symbol (if ({DriverType}:GetSuper Class)
                (get-abbrev
                    ({DriverType}:GetName
                        ({DriverType}:GetSuper Class))))
            ({DriverType}:GetName Class)))

(de :MustBeAMicroCeyxTupleType (Type)
    (if (typep Type '{TupleType}) Type
        (error DriverCurrentApiFunction 'DriverBadMicroCeyxClassType Type)))
```

({ObjectModelName}:ExistentClassP <ObjectModel> <Class>)
[SUBR à 2 arguments]

Méthode appelée pour tester si la classe représentée en DRIVER par l'objet système <Class> existe dans le modèle à objets.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:ExistentClassP (ObjectModel Class)
    (getfn1 (:GetMicroCeyxType Class) 'make))

(de :GetMicroCeyxType (Class)
    (symbol (if ({DriverType}:GetSuper Class)
                (get-abbrev
                    ({DriverType}:GetName
                        ({DriverType}:GetSuper Class))))
            ({DriverType}:GetName Class)))
```

4.2.1 Gestion des défauts d'objet

Les défauts d'objets dans le modèle sont gérés à travers les méthodes suivantes :

({ObjectModelName}:MakeObjectDefault <ObjectModel> <Class>)
[SUBR à 2 arguments]

Méthode appelée pour créer un défaut d'objet remplaçant une instance de la classe représentée en DRIVER par l'objet système <Class>.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:MakeObjectDefault (ObjectModel Class)
    (let ((Default (makevector 0 ())))
         (typevector Default (get-abbrev ({DriverType}:GetName Class)))
         Default))
```

({ObjectModelName}:ObjectDefaultP <ObjectModel> <Default>)
[SUBR à 2 arguments]

Méthode appelée pour tester si un objet est représenté en mémoire sous la forme d'un défaut d'objet.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:ObjectDefaultP (ObjectModel Default)
    (eqn (vlength Default) 0))
```

({ObjectModelName}:ChangeDefaultToObject
<ObjectModel> <Class> <Default>) **[SUBR à 3 arguments]**

Méthode appelée pour transformer un défaut d'objet en un objet de la classe <Class>.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:ChangeDefaultToObject (ObjectModel Class Default)
    (exchvector Default ({MicroCeyx}:MakeObject ObjectModel Class)))
```

({ObjectModelName}:ChangeObjectToDefault
<ObjectModel> <Class> <Object>) **[SUBR à 3 arguments]**

Méthode appelée pour transformer un objet en défaut d'objet de la classe <Class>.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:ChangeObjectToDefault (ObjectModel Class Object)
    (exchvector Object ({MicroCeyx}:MakeObjectDefault ObjectModel Class)))
```

4.2.2 Accès utilisateur aux objets

L'utilisateur qui souhaite accéder à ses objets (écriture comme lecture) alors qu'ils peuvent être représentés sous forme de défauts d'objet doit le faire par l'intermédiaire d'accesseurs particuliers, "protégés" que nous proposons de formuler sous forme de méthodes de modèles. Ces méthodes ne sont pas utilisées en interne à DRIVER ; elles existent uniquement à disposition de l'utilisateur final.

({ObjectModelName}:ProtectedReadSlotValue
<ObjectModel> <Slot> <Object>) **[SUBR à 3 arguments]**

Méthode appelée pour extraire d'un objet <Object> la valeur d'un slot <Slot>. Si l'objet est représenté par un défaut d'objet, on le charge auparavant. <Slot> est un objet système DRIVER de la classe `DriverSlot`.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:ProtectedReadSlotValue (ObjectModel Slot UserObject)
    (if ({MicroCeyx}:ObjectDefaultP ObjectModel UserObject)
        (DriverLoadObject UserObject))
    ({MicroCeyx}:ReadSlotValue ObjectModel Slot UserObject))

(de {MicroCeyx}:ReadSlotValue (ObjectModel Slot UserObject)
    (send ({DriverSlot}:GetName Slot) UserObject))
```

({ObjectModelName}:ProtectedWriteSlotValue
<ObjectModel> <Slot> <Object> <NewSlotValue>)
[SUBR à 4 arguments]

Méthode appelée pour affecter la valeur <NewSlotValue> dans le slot <Slot> de l'objet <Object>. Si l'objet est représenté par un défaut d'objet, on le charge auparavant. <Slot> est un objet système DRIVER de la classe `DriverSlot`.

Définition de la méthode pour "MicroCeyx" :

```
(de {MicroCeyx}:ProtectedWriteSlotValue
       (ObjectModel Slot UserObject NewSlotValue)
    (if ({MicroCeyx}:ObjectDefaultP ObjectModel UserObject)
```

```
        (DriverLoadObject UserObject))
    ({MicroCeyx}:WriteSlotValue ObjectModel Slot UserObject NewSlotValue))

(de {MicroCeyx}:WriteSlotValue (ObjectModel Slot UserObject NewSlotValue)
    (send ({DriverSlot}:GetName Slot) UserObject NewSlotValue))
```

5 Exemple d'application

5.1 Exemple de définition de bases de données

```
;;; begin of example.ll
;-------------------------
; Mapping creation
;-------------------------

(DriverUseDBMSStratification 'ObjectDriver 'NF2Driver 'ingres)
(DriverDefineDBMSDefaultObjectModel 'ObjectDriver 'MicroCeyx)
(DriverDefineUserStratification 'lebastard () 'lebastard () 'lebastar ())

(DriverDatabaseDefinitionMode 'demo 'demo 'smecidemo)

(unless (DriverExistentObjectTypeP 'symbol)
  (DriverCreateObjectPrimitiveType 'symbol 'string)
  (de Driver_string_symbol (str) (if str (symbol () str)))
  (de Driver_symbol_string (sym) (if sym (string sym))))


;-------------------------
; Relational database: Table definitions
;-------------------------

(DriverDefineRelationalTable emp
   (empno integer 2 KeyPart)
   (ename string 20 ())
   (fname string 20 ())
   (job   string 20 ())
   (mgr   integer 2 ())
   (sal   float   8 ())
   (com   float   8 ())
   (deptn integer 2 ()))

(DriverDefineRelationalTable dept
   (deptno integer 2 KeyPart)
   (dname  string 20 ()))
```

```
(DriverDefineRelationalTable deptsite
   (dept  integer 2 KeyPart)
   (sname string 20 KeyPart))


;---------------------------
; NF2 and object databases: Class definitions
;---------------------------

(DriverDefineClass employee
   (MainlyMappedOn emp)
   (TupleOf
       (symbol name
           (MappedOn (emp ename)))
       (symbol firstname
           (MappedOn (emp fname)))
       (symbol jobdesc
           (MappedOn (emp job)))
       (Atomic manager
           (ReferTo employee)
           (MappedOn (emp mgr)))
       (Atomic salary
           (MappedOn (emp sal))
           (RelConstraint
               (lambda (s) (and (>= s 700.) (<= s 9999.)))))
       (Atomic dept
           (MappedOn (emp deptn))
           (ReferTo department)
           (Inverse (department employees)))))

(DriverDefineClass salesman
   (SubclassOf employee)
   (TupleOf
       (symbol jobdesc
           (RelConstraint
               (lambda (a) (eq a 'salesman))))
       (Atomic commission
           (MappedOn (emp com)))
       (Computation income
```

```
           (DefinedBy (lambda (a b) (+ a b))
                      (emp sal) (emp com)))))

(DriverDefineClass department
   (MainlyMappedOn dept)
   (TupleOf
       (symbol name
           (MappedOn (dept dname)))
       (SetOf employees
           (MappedOn (e emp))
           (Atomic
               (MappedOn (e empno))
               (ReferTo employee))
           (ReadOnly t)
           (Inverse (employee dept))
           (JoinDescription dept e
                (lambda (a1 a2) (eq a1 a2))
                (dept deptno) (e deptn)))
       (ListOf sites
           (symbol sname
               (MappedOn (deptsite sname)))
           (MappedOn deptsite)
           (OrderedBy ((deptsite sname) Ascending)))
       (Computation salaryaverage
           (ComputationType MultiTupleComputation)
           (DefinedBy (lambda (a) (average a)) (emp sal))))
   (JoinDescription dept emp
        (lambda (a1 a2) (eq a1 a2))
        (dept deptno) (emp deptn))
   (JoinDescription dept deptsite
        (lambda (a1 a2) (eq a1 a2))
        (dept deptno) (deptsite dept)))

(DriverDatabaseDefinitionModeEnd)

;;; end of example.ll
```

5.2 Exemple de session de travail

```
$ lelisp
ILOG Le-Lisp 15.260, licensed to "inria-sophia  "
; Le-Lisp (by INRIA) version 15.26  (01/Dec/93)  [sun4]
; Complice : jeu  6 juin 96 17:16:25
= (31bitfloats abbrev callext compiler complice date debug defstruct format
hash-table lmcache loader messages microceyx module pathname pepe pretty
rehash setf stringio virbitmap virtty)
?  ^Ldriver
; Cloading ing_c.o
Chargement de DRIVER II
Chargement de DRIVER II Ok
DRIVER II v2.961105
Chargement des modeles a objets pour DRIVER
Chargement des modeles a objets pour DRIVER Ok
= driver.ll
? ^Lexample
= example.ll
? (DriverConnectLocalUser 'lebastard ())
= t
? (send 'Begin (DriverCreateTransaction 'demo))
= t
? (DriverCreateClasses)
= t
? ;;; Pour MicroCeyx
? (de #:tclass:prin (o)
?     (cond ((zerop (vlength o))
?            (prin "<An object default>"))
?           ((catcherror () (send 'name o))
?            (prin "<object " (send 'name o) ">"))
?           (t (prin "<An object>"))))
= #:tclass:prin
? (DriverQuery "select e from e in employee;")
= (<object martin> <object miller> <object james> <object ward> <object allen>
 <object jones> <object blake> <object clark> <object ford> <object scott>
 <object king>)
? (DriverQuery "select e.name from e in employee where e.salary>1200. and
 e.salary<2000.;")
```

```
= (ward martin james)
? (send 'Commit (DriverCurrentTransaction))
= t
? (end)
May Le-Lisp be with you.
$
```

6 Index des fonctions et des variables

Table des matières

Liste des derniers rapports de recherche du CERMICS

List of previous CERMICS research reports

96-74	B. Lapeyre Y. J. Xiao	*Volume-discrepancy of sequences and numerical tests.* 16 pages, septembre 1996, Noisy-Le-Grand
96-75	A. Toubol	*High temperature regime for a multidimensional Sherrigton-Kirkpatrick model of spin glass.* 34 pages, septembre 1996, Noisy-Le-Grand
96-76	C. Buisson., J. P.Lebacque, J. B.Lesort	*Travel Times Computation for Dynamic Assignment Modelling.* 15 pages, novembre 1996, Noisy-Le-Grand
96-77	E. Cancès, C.Le Bris	*On the perturbation methods for some nonlinear quantum chemistry models.* 45 pages, novembre 1996, Noisy-Le-Grand
96-78	S. Depeyre	*Une méthode couplée pour la simulation d'écoulements disphasiques dispersés.* 57 pages, octobre 1996, Sophia-Antipolis
96-79	N.Glinsky-Olivier J.F.Gerbeau B.Larrouturou	*Semi-implicit Roe-type fluxes for low-mach number flows.* ...pages, octobre 1996, Sophia-Antipolis
96-80	*B. Jourdain*	*Propagation du chaos trajectorielle pour les lois de conservation scalaire.* 14 pages, décembre 1996, Noisy-Le-Grand
96-81	*B. Jourdain*	*Diffusions with a nonlinear drift coefficient and probabilistic interpretation of generalized Burger's equations.* 16 pages, décembre 1996, Noisy-Le-Grand

96-82	*D. Hirschkoff*	*Up-to context proofs for the π-calculus in the coq system.* 18 pages, janvier 1997, Noisy-Le-Grand
96-83	*E. V Abrarova,* *A. V. Karapetyan*	*Sur la stabilité et les bifurcations des mouvements stationnaires d'un corps solide dans un champ de gravitation central.* 22 pages, janvier 1997, Noisy-Le-Grand
96-84	*E. V Abrarova,*	*Sur les mouvements stationnaires en orbite d'un système de deux corps avec liaison élastique.* 20 pages, janvier 1997, Noisy-Le-Grand
97-85	J.P. Cioni	*Parallelisation of Maxwell and 3D simulations in electromagnetism using clusters of workstations.* 18 pages, janvier 1997, Sophia-Antipolis
97-86	B. Neveu G. Trombettoni	*Computational complexity of Multi-way, Dataflow constraint problems.* 13 pages, janvier 1997, Sophia-Antipolis
97-87	F. Lebastard	*Driver II (Lisp): Manuel de référence.* 52 pages, janvier 1997, Sophia-Antipolis

Ces rapports peuvent être obtenus en s'adressant au secrétariat du CERMICS :
The reports can be asked from:

Imane HAMADE
ENPC-CERMICS
6 et 8 Avenue Blaise Pascal
Cité Descartes Champs-sur-Marne
77455-Marne-La-Vallée CEDEX 2
Tél : (33) 01 - 64-15-35-71
email: `hamade@newaphro.enpc.fr`