TR-87-39

Defining Views in the Binary Relationship Model

by

Leo Mark

Defining Views in the Binary Relationship Model

Leo Mark
Department of Computer Science
University of Maryland
College Park, Md. 20742

Submitted February 1986
Revised October 1986

The Binary Relationship Model has been praised for the way it not only supports, but practically forces us to model the "deep structures" of the information in the conceptual schema. An adverse effect is that the conceptual schema becomes very large. This, combined with the fact that a Binary Relationship Model schema gives a "flat" representation of the information, makes it very hard to distinguish important concepts of a model from its less important details. Furthermore, the length of DML statements is directly proportional to the size of the schema. What we need for practical applications is a way of keeping the "deep structures" while seing only the "surface structures". We need to be able to define views on a conceptual schema. We need to be able to select and derive from the conceptual schema precisely the concepts we are interested in for each application of our model.

We propose two new concepts for view definition in the Binary Relationship Model, **DOTs and shortcuts.** DOTs are derived object types and shortcut are derived roles which are used in the definition of derived binary relationship types.

1. History & Background

The Binary Relationship Model is one of the many data models that appeared in the mid 70's. The two basic modeling concepts are object types and binary relationship types between them. Abrial's Semantic Binary Data Model [Abrial 74] is to our best knowledge the first proposed Binary Relationship Model. Senko's Data Independent Accessing Model (DIAM) [Senko 75] distinguishes two kinds of object types, entity sets and entity name sets. It also comprises a data manipulation language, FORAL, that later appears in an elegant light-pen version [Senko 78]. Hall, Owlett, and Todd introduce the notion of surrogates [Hall, Owlett, and Todd 76] to represent entity sets. Nijssen, one of the early and strong promoters of the Binary Relationship Model, makes several contributions [Nijssen 77, 79, 81]. Kent strongly argues for the Binary Relationship Model as a means for representing irreducible facts in [Kent 78]. The notion of "IS-A" relationships [Smith and Smith 77] is supported in the Binary Relationship Model in

Research supported in part by the Systems Research Center, University of Maryland under the National Science Foundation Grant OIR-85-00108.

[Ceri, Pelagatti, and Bracchi 81] and in [ISO 82]. Buneman and Frankel [Buneman and Frankel 79] introduces a purely functional data manipulation language, FQL. A more user-friendly functional data manipulation language, DAPLEX, with the possibility to define derived concepts is presented by Shipman in [Shipman 81]. Several types of constraints are developed for the Binary Relationship Model by Breutman, Falkenberg, and Mauer in [Breutman, Falkenberg, and Mauer 79], by Nijssen in [Nijssen 81], and in [ISO 81]. A user-friendly data manipulation language, RIDL, is presented by Meersman in [Meersman 81]. Vermeir presents a schema abstraction tool for the Binary Relationship Model in [Vermeir 83]. It produces a hierarchy of abstractions of a schema relative to a given viewpoint of interest in the schema. The abstractions consist of parts of the schema selected on the basis of their closeness and degree of connection to a given viewpoint. Some of our previous work on the Binary Relationship Model include [Mark 83] and [Mark and Roussopoulos 83, 85]. In [Mark 83] we raise several questions that unveil problems with the Binary Relationship Model and we propose solutions to some of them. We also define a self-describing meta-schema for the Binary Relationship Model. In [Mark and Roussopoulos 83] we propose a data manipulation language that together with the self-describing meta-schema allows manipulation of all data and data descriptions in a database. In [Roussopoulos and Mark 85] we concentrate on aspects of schema manipulation using this data manipulation language.

In section 2 of this paper we first give a brief presentation of the graphical formalism used for schema design in the Binary Relationship Model. As an example of its use we design a simple schema which will be used as a basis for most of the examples in this paper. We also give a brief presentation of a data manipulation language for the Binary Relationship Model.

After this introductory section, we proceed in section 3 with an informal presentation of two new concepts for view definition in the Binary Relationship Model, DOTs and shortcuts. We provide several examples based on the simple schema presented in section 2.

Section 4 formally defines the two new concepts. After a simple set of renaming rules we design a meta-schema for the Binary Relationship Model in terms of itself. This meta-schema is essential for the following formal definition of DOTs and shortcuts. Having formally defined the new concepts, we incorporate them in the meta-schema thereby making them part of the Binary Relationship Model. The examples in this section are based on the meta-schema. They demonstrate how the concepts can be used for meta-data management and how the concepts can actually be used for presenting only the "surface structures" of the data model. We end the paper by listing a set of simple rules for view update in the Binary Relationship Model extended with the view concepts. These rules correspond to similar rules in the relational data model.

2. The Binary Relationship Model

2.1 Concepts and Graphical Formalism

The following symbols are the building blocks of a conceptual schema in the Binary Relationship Model [Nijssen 81], [ISO 82].

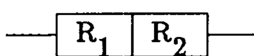
Linguistic object types are represented by circles with a name inside.

A lexical object type (LOT) with name 'LOT NAME' represents a class of strings used to name objects. For example, NAME is a LOT having as elements {'SMITH', 'MARK', 'JONES',...}; TITLE has as elements {'DATA', 'ADA', 'CLU', 'NORMALIZATION',...}.

A non-lexical object type (NOLOT) with name 'NOLOT NAME' represents a class of real world entities each of which is represented in the system by a surrogate [Hall, Owlett, Todd 76]. There is a one-to-one correspondence between surrogates and real world entities. Surrogates are distinct and generated by the system. They are invisible to and not changeable by the user of the system.

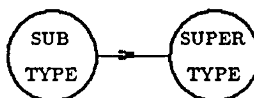
The distinction between lexical and non-lexical object types makes it possible to represent an object separately from the lexical objects naming it.

Binary relationship types are represented by a box, and two names inside.



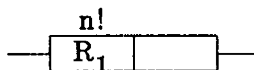
A binary relationship type is an ordered pair of LOTs and/or NOLOTs; it represents a relation between the two linguistic types. 'R1' and 'R2' are names for the ROLEs played by the linguistic object types in the binary relationship type. An example, of a relationship is CAR-OWNSHIP which relates the NOLOTs PERSON and CAR with roles OWNER and OWNEE respectively.

The relationship type defines a relation between the sets of occurrences of the two object types. We use the term active set for role 'R' to represent the set of objects actually appearing in the tuples of the relation from the object type in role 'R'. The set of occurrences of an object type is equal to the union of all active sets for the object type.

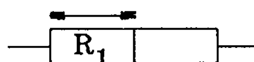


This special symbol for a binary relationsip type - called an IS-A or SUB-TYPE LINK - is used whenever a NOLOT IS-A subtype of another NOLOT or a LOT IS-A subtype of another LOT. An IS-A relationship defines a total function from the set of occurrences of the sub-object type to the set of occurrences of the super-object type.

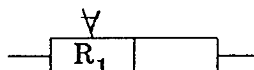
The following symbols can be used to constrain relationships in the database:



n! Cardinality constraint. The symbol constrains the relation to have exactly n tuples for each element in the active set for role 'R1'.

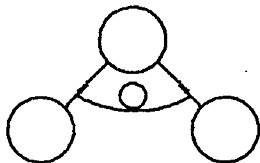


↔ Unique constraint. The symbol contraints the relation to be a (partial) function. This is a special case of the cardinality constraint with n=1.



∀ Total constraint. The symbol constrains the relation to be total, i.e. the active set for the object type in role 'R1' should be equal to the union of all active sets of the object type. The symbol can be used separately, together with the cardinality constraint symbol, or together with the unique constraint symbol.

The symbol ○ defines a constraint on the sets of occurrences of object types in the following cases:



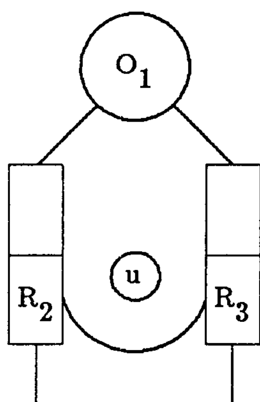
Ⓧ Exclusion between the sets of occurrences of subtypes, or between active sets of an object type.

Ⓟ Partition between the sets of occurrences of subtypes, or between active sets of an object type.

⊂ Subset between two active sets of an object type.

⊜ Equality between two active sets of an object type.

There are no symbols for defining subset or equality between two relations on the same two object types.



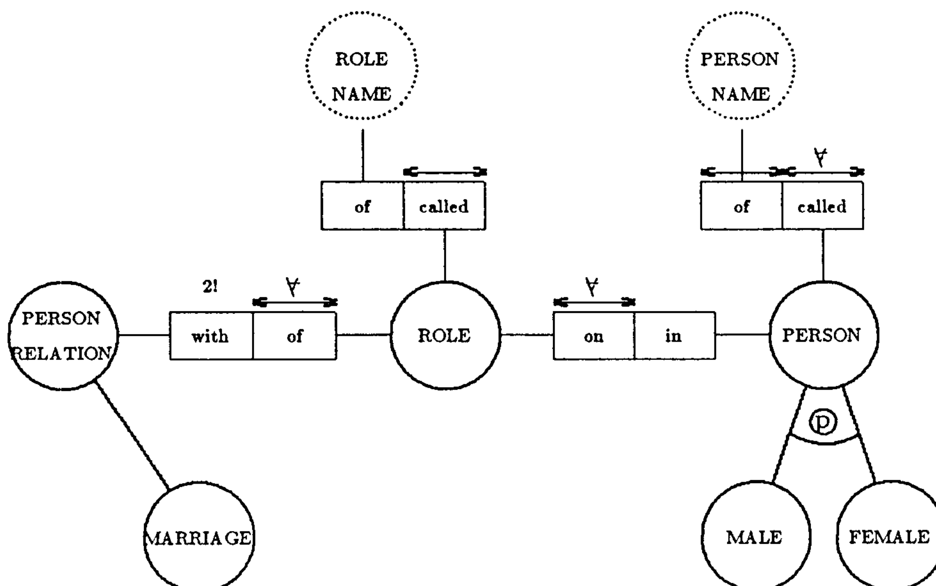
Ⓤ Uniqueness. The symbol defines a partial function from the Cartesian product of the active sets involved to the set of occurrences of an object type. This object type must be different from those of the active sets involved and relationship types must be directly defined between this object type and each of the object type of the active sets involved. E.g. in the example to the left the partial function is from the Cartesian product of the active sets in roles R_2 and R_3 to the active set of the object type O_1.

This symbol combined with other constraint symbols has a variety of uses, e.g. for representing n-ary relations with composite keys (in the relational model sense.)

2.2 The Schema

We illustrate the use of the graphic formalism for schema definition in figure 1.

Figure 1. Example of a schema



The schema can be read as follows. It defines PERSON_RELATIONs 'with' precisely two (2!) ROLEs. Each ROLE can be 'called' at most (↔) one ROLE_NAME. Each (∀) ROLE is 'of' at most one (↔) PERSON_RELATION and 'on' at most one (↔) PERSON, each (∀) of which is in turn uniquely (↔) named by an PERSON_NAME. PERSON_NAMEs are 'of' at most one (↔) PERSON. PERSONs are 'partitioned' ⓟ into MALEs and FEMALEs. MARRIAGEs are PERSON_RELATIONs 'with' ROLE 'called' 'HUSBAND_OF' 'on' MALEs and 'with' ROLE 'called' 'WIFE_OF' 'on' FEMALEs.

2.3 The Data Manipulation Language

The DML language briefly overviewed here is network-like and was presented in [Mark and Roussopoulos 83]. It allows the user to select, traverse, connect and disconnect occurrences of object types. It also allows the user to list occurrences of lexical object types and to create new surrogates as occurrences of non-lexical object types. The four basic commands are list, connect, disconnect, and reconnect.

The list command has the following structure:

list S;

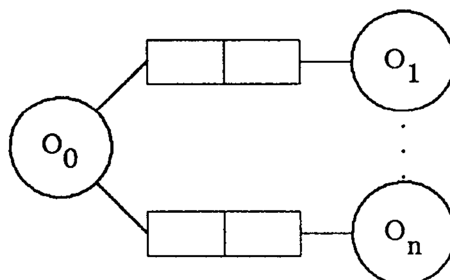
where S is a set of occurrences of a lexical object type, O. The set S can recursively be qualified by conditions on the extensions of the binary relationship types, the object type O is a part of. We shall use the list command in numerous examples in the next section.

The skeletons for the connect, disconnect, and reconnect commands are:

connect S_0	**disconnect** S_0	**reconnect** S_0
to S_1	**from** S_1	**from** S_1
.	.	.
.	.	.
.	.	to S_i
to S_n	**from** S_n	.

where S_0, S_1,...,S_n are occurrences of the object types O_0, O_1,..., O_n shown in figure 2. If the binary relationships are un-constrained, then the connect command inserts tuples of the form (S_0,S_i) into the corresponding binary relationship (O_0,O_i). The disconnect command removes tuples (S_0,S_i) from the corresponding binary relationships (O_0,O_i). Note that $S_0,S_1,...,S_n$ may be sets of occurrences rather than single objects. In this case, the result of the connect command is the union of the Cartesian product, $S_0 \times S_i$, and the set of occurrences of the binary relationship type between O_0 and O_i, (i.e., extension$(O_0,O_i) \cup (S_0 \times S_i)$), for i = 1,...,n. The result of the disconnect command is the set difference between the set of occurrences of the binary relationship type between O_0 and O_i and the Cartesian product of S_0 and S_i (i.e., extension(O_0,O_i) - $(S_0 \times S_i)$), for i = 1,...,n. Note also that in this case of sets of occurrence the two operations are not each other's inverse.)

Figure 2. Simple connection path.



The reconnect command is often equivalent to disconnect S_0 from S_1 later followed by connect S_0 to S_i. However, the hybrid is still needed because we may not be able to identify the set S_0 for the connect command once S_0 has been disconnected from S_1. The scope of S_0 is one statement. Furthermore, any of the two statements may violate an integrity constraint. The unit of a transaction is one DML-statement. Notice, that the last of these problems would still exist if we connect in the first statement and disconnect in the second.

EXAMPLE 1

To insert information about two new people, Liz and Ed, in the database we use the statements:

```
connect new FEMALE
        to PERSON_NAME: 'LIZ'.

connect new MALE
        to PERSON_NAME: 'ED'.
```

□

The primitive, **new**, simply produces a unique new surrogate of the non-lexical object type following it. If this surrogate is the member of a subtype, then it also produces a new unique surrogate of the supertype(s), and connects the two surrogates in the subtype-link. A subtype-link maintains in other words a "same" relationship between sets of surrogates.

Transactions are handled by nested commands. The skeleton for nesting the connect commands is illustrated by the following example.

Each level in a nesting returns the head node of the binary relationship which can then be used by the previous level. Constraints on the database are checked after the transaction; if they are violated the transaction is undone. For each type of constraint in the data model the DML-processor has a procedure capable of checking it.

EXAMPLE 2

To insert the fact that Liz and Ed just got married we use the statement:

```
connect new MARRIAGE
    to (connect new ROLE
            to ROLE_NAME: 'WIFE_OF'
            to FEMALE called 'LIZ')
    to (connect new ROLE
            to ROLE_NAME: 'HUSBAND_OF'
            to MALE called 'ED').
```

□

Examples of the language will be used throughout the paper. For a more detailed presentation of the language, please see [Mark and Roussopoulos 83].

3. Views - Meet the Concepts

The Binary Relationship Model has been praised for the way it not only supports, but practically forces us to model the "deep structures" of the information in the conceptual schema

[Nijssen 81]. An adverse effect is that the conceptual schema becomes very large. This, combined with the fact that a Binary Relationship Model schema gives a "flat" representation of the information, makes it very hard to distinguish important concepts of a model from its less important details. Furthermore, the length of DML statements is directly proportional to the size of the schema. What we need for practical applications is a way of keeping the "deep structures" while seing only the "surface structures". We need to be able to define views on a conceptual schema. We need to be able to select and derive from the conceptual schema precisely the concepts we are interested in for each application of our model.

3.1 Derived Object Types - DOTs

Let us first illustrate the notion of a derived object type - a DOT.

EXAMPLE 3

The following simple DML statement:

list PERSON_NAME of MALE in ROLE of MARRIAGE

will list the names of all persons that are husbands in marriages.

If we introduce a new object type, 'HUBBY', which is an object type derived from 'MALE' with the deriving expression being:

MALE in ROLE of MARRIAGE

then the above DML statement can be written:

list PERSON_NAME of HUBBY.

The DML interpreter will evaluate the DML statement by simply substituting the name of the derived object type 'HUBBY' by the deriving expression for it.

Since 'HUBBY' is derived from 'MALE' it is natural that it inherits some of the properties of 'MALE' which in turn inherits its properties from 'PERSON', e.g. an element of 'HUBBY' is 'called' some PERSON_NAME, and it is 'in' some ROLE.

□

EXAMPLE 4

The deriving expressions for the derived object types 'MALE_NAME' and 'FEMALE_NAME' from 'PERSON_NAME' are as follows:

PERSON_NAME of MALE

PERSON_NAME of FEMALE

□

EXAMPLE 5

Similarly, the derived object types 'MALE_FEMALE_RELATIONSHIP', 'MALE_RELATIONSHIP', and 'FEMALE_RELATIONSHIP' from 'PERSON_RELATION' have the following deriving expressions:

PERSON_RELATION (with ROLE on MALE **and** with ROLE on FEMALE)

PERSON_RELATION ¬(with ROLE on FEMALE)

PERSON_RELATION ¬(with ROLE on MALE)

□

3.2 Derived Binary Relationship Types - Shortcuts.

Let us then illustrate the notion of a **derived binary relationship type** - a shortcut.

EXAMPLE 6

The DML statement:

list PERSON_NAME of PERSON in ROLE of PERSON_RELATION.

will simply list the names of all persons which are part of some person relationship.

If we introduce one of the roles, 'in', of a new binary relationship type derived from those already present using the following deriving expression for 'in':

of PERSON in ROLE of

then we can use this shortcut to simplify the above DML statement to the following:

list PERSON_NAME in PERSON_RELATION.

Again, the DML interpreter will evaluate the expression by substituting the name of the shortcut with its deriving expression.

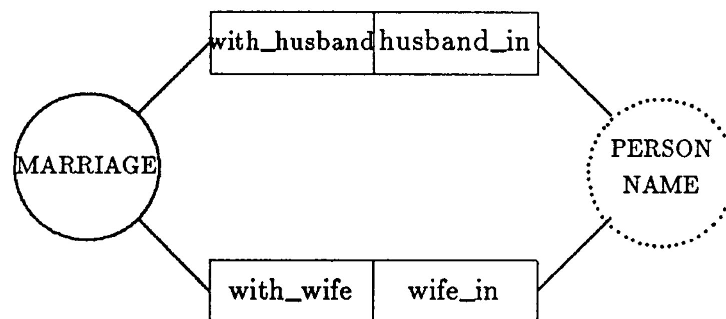
□

Nothing prevents us from using derived object types and shortcuts in the definition of new ones. This gives us a very powerful tool for view definition - similar to the one for the relational model, but with the characteristics of the Binary Relationship Model.

EXAMPLE 7

Figure 3 illustrates a possible view on the schema obtained by using derived object types and derived binary relationship types.

Figure 3. A view on the Schema.



To help us define the view we first define the following two derived object types from the object type 'ROLE':

MAN_ROLE: ROLE called ROLE_NAME='HUSBAND_OF'

WOMAN_ROLE: ROLE called ROLE_NAME='WIFE_OF'

We note, that the notation used only indicates the components of the view. Later in the paper we discuss how derived object types, derived binary relationship types, and views are actually defined.

To complete the view definition, we continue as follows:

with_husband: with MAN_ROLE on PERSON called

husband_in: of PERSON in MAN_ROLE of

with_wife: with WOMAN_ROLE on PERSON called

wife_in: of PERSON in WOMAN_ROLE of

□

3.3 Composed Binary Relationship Types - Shortcuts

The **derived** binary relationship types above have roles that follow simple and inverse paths. One simple path out **same** simple path home!

Are there **composed** binary relationship types with an extension which is the **intersection** of the extensions of two or more alternative paths? One composed path out **same** composed path home! Are there **composed** binary relationship types with an extension which is the **set difference** of the extensions of two alternative paths? One composed path out **same** composed path home! Are there **composed** binary relationship types with an extension which is the **union** of the extensions of two alternative paths? One composed path out **same** composed path home! Are there **composed** binary relationship types with two derived roles which do **not** follow inverse paths? One path out **another** path home!

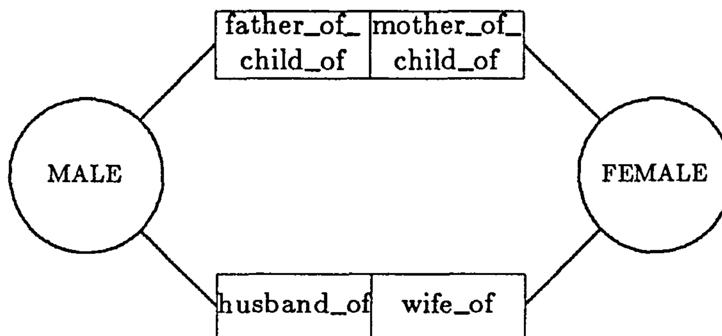
The answer to all, except the last, of these questions is yes. We shall discuss each question in the following examples.

EXAMPLE 8

Intersection:

male_role: father_of_child_of **and** husband_of
female_role: mother_of_child_of **and** wife_of

Figure 4. Composed path



We could not have defined this composed shortcut using DOTs and avoiding the conjunction of role names:

FATHER_AND_HUSBAND: MALE (father_of_child_of FEMALE **and** husband_of FEMALE)

MOTHER_AND_WIFE: FEMALE (mother_of_child_of MALE **and** wife_of MALE)

is_father_of_child_of_and_husband_of: father_of_child_of

is_mother_of_child_of_and_wife_of: mother_of_child_of

The mistake is that the DOT, 'FATHER_AND_HUSBAND', includes all males that are fathers and husbands, but not necessarily of the children of the same female as they are husbands of.

□

EXAMPLE 9

Set Difference:

male_role: father_of_child_of **and** ¬husband_of
female_role: mother_of_child_of **and** ¬wife_of

We could not have defined this composed Shortcut using DOTs and without the conjunction above:

FATHER_NOT_HUSBAND: MALE (father_of_child_of FEMALE **and** ¬husband_of FEMALE)

MOTHER_NOT_WIFE: FEMALE (mother_of_child_of MALE **and** ¬wife_of MALE)

is_father_of_child_of_not_husband_of: father_of_child_of

is_mother_of_child_of_not_wife_of: mother_of_child_of

The mistake is that, e.g. the DOT, 'FATHER_NOT_HUSBAND', includes all males that are fathers and not husbands to anyone, whereas what we want is the fathers who are not husbands of their childrens' mother.

□

EXAMPLE 10

Union:

male_role: father_of_child_of **or** husband_of
female_role: mother_of_child_of **or** wife_of

We could not have defined this composed shortcut using DOTs and avoiding the disjunction of role names:

FATHER_OR_HUSBAND: MALE ¬(¬(father_of_child_of FEMALE) **and** ¬(husband_of FEMALE))

MOTHER_OR_WIFE: FEMALE ¬(¬(mother_of_child_of MALE) **and** ¬(wife_of MALE))

The problem is that the roles we have available for the Shortcut definition will not map one whole DOT to the other, e.g. father_of_child_of does not map all of FATHER_OR_HUSBAND to MOTHER_OR_WIFE.

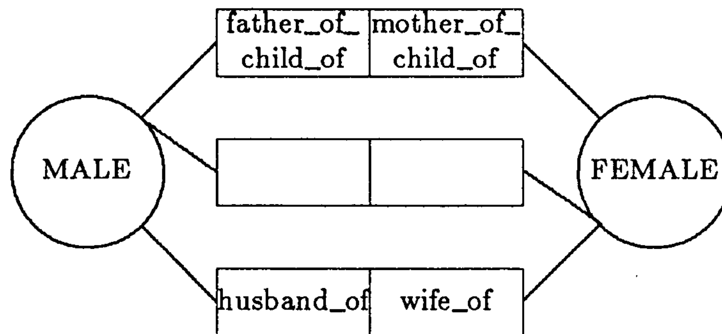
□

EXAMPLE 11

One path out another path home:

male_role: father_of_child_of
female_role: wife_of

Figure 5. One path out another path home.



If (x,y) is in the extension of this composed Shortcut then x is the father_of_child_of y and y is the wife_of x. This implies, that y is the mother_of_child_of x and x is the husband_of y.

The **extension** of this composed Shortcut is the **intersection** of the two involved binary relationship types.

We do NOT allow this kind of definition of composed binary relationship types! All composed binary relationship types are defined with roles that follow inverse paths!

□

We emphasize, that a role deriving expression does not nescessarily by itself uniquely identify the role(s) it is composed from. In the following sections we shall explain how to avoid ambiguity.

We finally note, that roles of subtype-links can be part of role deriving expressions like any other role of a binary relationship type. This is however rarely used because subtypes inherit the binary relationship types of their supertypes; but, if we want to define a derived role for some subtypes without making it available for their supertype, then we need it.

4. Views - Formal Definition

In this section we provide a formal definition the two new concepts. After a simple set of renaming rules we design a meta-schema for the Binary Relationship Model in terms of itself. This meta-schema is essential for the following formal definition of DOT and shortcuts. Having formally defined the new concepts, we incorporate them in the meta-schema thereby making them part of the Binary Relationship Model. The examples in this section are based on the meta-schema. They demonstrate how the concepts can be used for meta-data management and how the concepts can actually be used for presenting only the "surface structures" of the data model. We end this section by listing a set of simple rules for view update in the Binary Relationship Model extended with the view concepts. These rules correspond to similar rules in the relational data model.

4.1 Renaming Rules, etc.

The following set of rules for renaming, omission, etc. applies to the definition of a view:

- An object type can be renamed. Object type names must be unique within each view. The rule applies to the names of all object types in a view, including the derived object types.

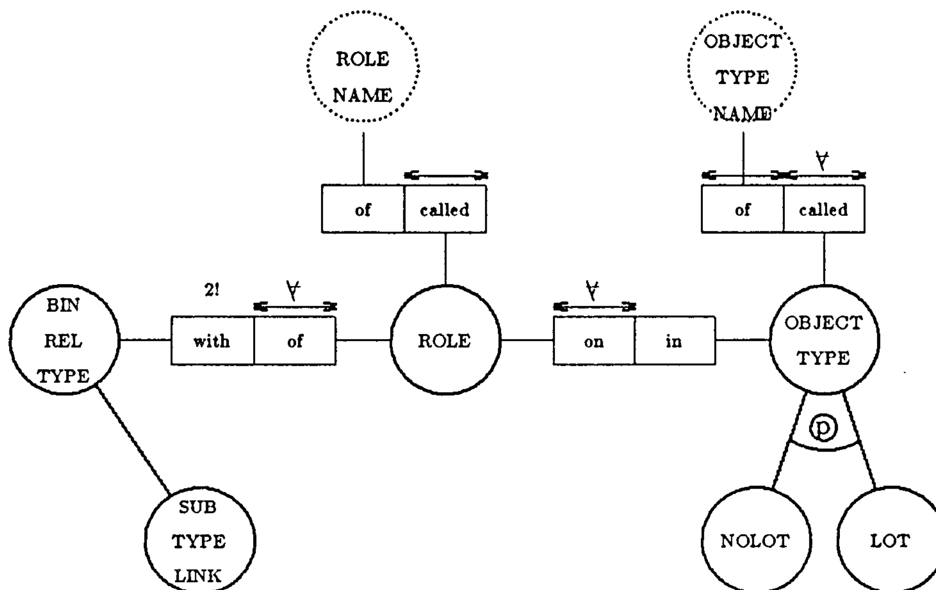
- A role can be renamed. If two object types are directly connected by more than one binary relationship type in a view, then the role names of these binary relationship types must be unique for each object type. The rule also applies if one or more of these binary relationship types are derived.
- Any object type may be omitted in a view definition. If an object type is omitted in a view definition, so are all binary relationship types, including subtype links, the object type is part of. If the omitted object type has any subtypes or derived object types, then the binary relationship types inherited by these will not be affected.
- Any binary relationship type may be omitted from a view definition.
- Any constraint may be omitted from the view. They will of course still exist and be enforced at the conceptual schema level.

Renaming an object type or a role is just a simple case of defining derived object types or roles.

4.2 The Meta-Schema

To be able to give precise definitions of the two new concepts, derived object types and derived binary relationship types, we need a precise definition of the basic concepts of the data model. We give this definition in terms of the data model itself by designing a meta-schema for it.

Figure 6. The Meta-Schema



The meta-schema can be read as follows. It defines BIN_REL_TYPEs 'with' precisely two (2!) ROLEs. Each ROLE can be 'called' at most (↔) one ROLE_NAME. Each (∀) ROLE is 'of' at most one (↔) BIN_REL_TYPE and 'on' at most one (↔) OBJECT_TYPE, each (∀) of which is in turn uniquely (↔) named by an OBJECT_TYPE_NAME. OBJECT_TYPE_NAMEs are 'of' at most one (↔) OBJECT_TYPE. OBJECT_TYPEs are 'partitioned' (p) into lexical- and non-lexical object types (LOTs and NOLOTs). SUBTYPE_LINKs are BIN_REL_TYPEs 'with' ROLE 'called' 'SUB' 'on' one of the OBJECT_TYPEs and 'with' ROLE 'called' 'SUP' 'on' the other OBJECT_TYPE. (Only SUBTYPE_LINKS will be allowed to have role names, 'SUB' and 'SUP'.

We shall not here concentrate on the details of making a meta-schema self-describing. For a detailed discussion see [Mark 85].

In the extension of the meta-schema we store both a description of application schemata defined in terms of the Binary Relationship Model and a description of the meta-schema itself. This latter description can be used whenever we want to retrieve information about the general rules and laws for defining schemata using the Binary Relationship Model since this is exactly what the meta-schema describes. We obviously do not want to change the stored description of the meta-schema.

4.3 DOTs

A derived object type is an object type with an object type name and an object type deriving expression. Using curly brackets for repetition, square brackets for optionality, slash for choice, bold types for terminals, and less than and greater than brackets for non-terminals, the syntax for object type deriving expressions is:

<dot_expression>	::=	<nolot_object_set>
	/	<lot_object_set>
<nolot_object_set>	::=	<nolot_name>
	/	<nolot_name><ref>
	/	<nolot_name>(<ref> {<cont>}})
<lot_object_set>	::=	<lot_name>[<comp><value>]
	/	<lot_name><ref>
	/	<lot_name>(<ref> {<cont>})
<cont>	::=	**and** <ref>
	/	**or** <ref>
<ref>	::=	¬(<ref>)
	/	<role_name><nolot_object_set>
	/	<role_name><lot_object_set>
<comp>	::=	=/</>/≤/≥/≠

The syntax for derived object type expressions is very similar to the syntax for the **list** statement, which is

<list_statement> ::= **list** <lot_object_set>

Note, that the syntax for derived object type expressions does not specify how we actually define a derived object type only what one of the ingredients, a derived object type expression, looks like.

For brevity, <value>, <role_name>, <lot_name>, and <nolot_name> are not defined in further detail. A <value> will be represented by a string of digits or upper case letters enclosed in single quotes. A <role_name> will be represented by a lower case string. A <nolot_name> will be represented by an upper case string. A <lot_name> will be represented by an upper

case string.

Only a small subset of the object type deriving expressions following this syntax defines a derived object type for a given schema. An object type deriving expression must also obey the following context sensitive rules. A <lot_name> must be a stored value of the object type, 'OBJECT_TYPE_NAME', and this value must be connected to an occurrence of the object type, 'LOT'. A <nolot_name> must be a stored value of the object type, 'NOLOT_NAME', and this value must be connected to an occurrence of the object type, 'NOLOT'. A <role_name> must correspond to a stored value of the object type, 'ROLE_NAME'. Alternatively, we could have defined <lot_name> and <nolot_name> as values of two LOTs, 'LOT_NAME' and 'NOLOT_NAME', respectively, which would then have to be explicitly modeled in the meta-schema.

In addition, an object type deriving expression must "match" the given schema. An object type deriving expression matches a schema if and only if it obeys the following set of rules. The <value> of a <lot_object_set> must have the type used to represent the LOT identified by the <lot_name> of the <lot_object_set>. The first <role_name> in a <ref> of a <lot_object_set> or a <nolot_object_set> must be the name of a role for the LOT or NOLOT preceding it. We note, that subtypes inherit the roles of their supertypes. The first <role_name> of all <ref>s in a conjunction/disjunction of <ref>s must be the name of some role for the LOT or NOLOT preceding the conjunction/disjunction of <ref>s. The first <nolot_name> in a <nolot_object_set> of a <ref> must be the name of a NOLOT which is the range of the role represented by the <role_name> of the <ref>. The first <lot_name> in a <lot_object_set> of a <ref> must be the name of a LOT which is the range of the role represented by the <role_name> of the <ref>.

The semantics of an object type deriving expression is as follows. A <nolot_object_set> defines a subset of the set of occurrences of the NOLOT represented by <nolot_name>. If the <nolot_object_set> consists of <nolot_name> only, then the two sets are identical. If the <nolot_object_set> consists of <nolot_name> followed by <ref>, then the subset is identical to the particular active set of the NOLOT in the role represented by <role_name> of <ref> which is connected to the <nolot_object_set> or <lot_object_set> following the <role_name> in <ref>. If the <nolot_object_set> consists of <nolot_name> followed by a conjunction/disjunction of <ref>s, then the subset is identical to the particular active set of the NOLOT, which fulfils the above requirement for the conjunct/disjunct of the <ref>s. If a <ref> is negated, then the subset is identical to the particular active set of the NOLOT, which fulfils the above requirements for the other <ref>s, but which has no elements connected to the <nolot_object_set> or <lot_object_set> of the negated <ref> via the role represented by the <role_name> of the negated <ref>.

A <lot_object_set> defines a subset of the set of occurrences of the LOT represented by <lot_name>. If the <lot_object_set> consists of <lot_name> only, then the two sets are identical. If the <lot_object_set> consists of <lot_name> followed by <comp> and <value>, then the subset is identical to the particular subset of the active set of the LOT whose elements are related by <comp> to <value>. If the <lot_object_set> consists of <lot_name> followed by <ref>, then the subset is identical to the particular active set of the LOT in the role represented by <role_name> of <ref> which is connected to the <nolot_object_set> or <lot_object_set> following the <role_name> in <ref>. If the <lot_object_set> consists of <lot_name> followed by a conjunction/disjunction of <ref>s, then the subset is identical to the particular active set of the LOT, which fulfils the above requirement for conjunct/disjunct of the <ref>s. If a <ref> is negated, then the subset is identical to the particular active set of

the LOT, which fulfils the above requirements for the other <ref>s, but which has no elements connected to the <nolot_object_set> or <lot_object_set> of the negated <ref> via the role represented by the <role_name> of the negated <ref>.

Derived object types inherit the binary relationship types of the object types they are derived from.

EXAMPLE 12

Using this syntax the deriving expressions for the derived object types 'LOT_NAME' and 'NOLOT_NAME' from 'OBJECT_TYPE_NAME' are as follows:

OBJECT_TYPE_NAME of LOT

OBJECT_TYPE_NAME of NOLOT

□

EXAMPLE 13

Similarly the deriving expressions for the derived object types 'BRIDGE', 'IDEA', and 'PHRASE' [Nijssen 81] from 'BIN_REL_TYPE' are as follows:

BIN_REL_TYPE (with ROLE on LOT and with ROLE on NOLOT)

BIN_REL_TYPE ¬(with ROLE on LOT)

BIN_REL_TYPE ¬(with ROLE on NOLOT)

□

EXAMPLE 14

The object type, 'META_LOTS', derived from 'LOT' and consisting of the lexical object types in the meta-schema has the following object type deriving expression:

LOT(called OBJECT_TYPE_NAME='OBJECT_TYPE_NAME' or called OBJECT_TYPE_NAME='ROLE_NAME')

□

EXAMPLE 15

We may want to define the derived object types, 'APPLICATION_LOT' and 'APPLICATION_NOLOT' consisting of all lexical and non-lexical object types, respectively, that are not in the meta-schema. The deriving expression for 'APPLICATION_LOT' would be:

LOT (called OBJECT_TYPE_NAME ≠ 'OBJECT_TYPE_NAME' and called OBJECT_TYPE_NAME ≠ 'ROLE_NAME')

□

EXAMPLE 16

We can use derived object types in the definition of other derived object types. As an example, we can define the derived object type 'APPLICATION_BRIDGE' from the derived object types 'BRIDGE', 'APPLICATION_LOT', and 'APPLICATION_NOLOT'. The deriving expression is:

BRIDGE (with ROLE on APPLICATION_LOT and with ROLE on APPLICATION_NOLOT)

□

4.4 Shortcuts; Derived and Composed

A derived binary relationship type is a relationship type with derived roles. The syntax for role deriving expressions is:

<shortcut_expression>	::=	<role_name>
		{<object_type_name><role_name>}
	/	<fork>
<fork>	::=	(<fork>)
	/	<fork> and <fork>
	/	<fork> or <fork>
	/	<role_name>
	/	¬<role_name>
<object_type_name>	::=	<nolot_name>
	/	<lot_name>

We could without problems have replaced the non-terminal, <object_type_name>, with the non-terminal, <dot_expression>, from the definition of the syntax for object type deriving expressions to obtain a more powerful tool for directly defining derived roles. The resulting role deriving expressions could however become quite complicated with such a syntax, which is why we force the user to explicitly define the derived object types needed in a role deriving expression before it is defined.

We could also easily have replaced the non-terminal, <fork>, by something more general. As it is now, shortcuts that serially connect roles can be arbitrarily long, whereas shortcuts that parallelly connect roles can only have length one. This forces the user, who wants to parallelly connect long "paths" between two object types, to first define each of these "paths" serially as shortcuts, and based on those define the shortcut that parallelly connects the two "paths". Again, to keep our minds clear, we have chosen the simple syntax.

EXAMPLE 17

Figure 7 illustrates a possible view on the meta-schema obtained by using derived object types and derived binary relationship types. To help us define the view we first define the following two derived object types from the object type 'ROLE':

SUB_ROLE: ROLE called ROLE_NAME='SUB'

SUP_ROLE: ROLE called ROLE_NAME='SUP'

We now define the view to consist of two derived object types and two derived binary relationship types connecting them. We note, that the notation used below only indicates the components of the view. Later in the paper we discuss how derived object types, derived binary relationship types, and views are actually defined.

DOMAIN_NAME: OBJECT_TYPE_NAME

IS_A: SUBTYPE_LINK

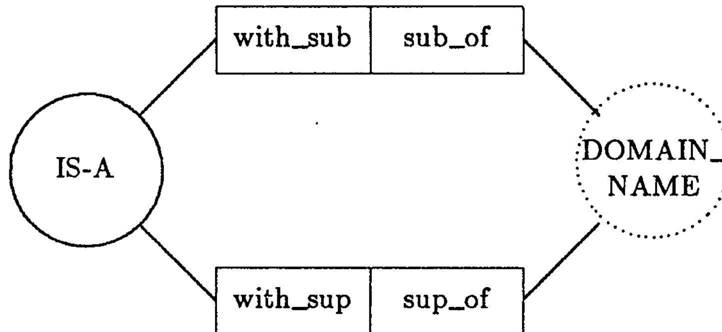
with_sub: with SUB_ROLE on OBJECT_TYPE called

sub_of: of OBJECT_TYPE in SUB_ROLE of

with_sup: with SUP_ROLE on OBJECT_TYPE called

sup_of: of OBJECT_TYPE in SUP_ROLE of

Figure 7. A view on the meta-schema



We can now, as an example, list all the "roots" of a subtype hierarchy as follows:

list DOMAIN_NAME(sup_of IS_A **and** ¬(sub_of IS_A))

□

The representation of <role_name>, <nolot_name>, and <lot_name> is the same one used in the syntax for object type deriving expressions.

Only a small subset of the role deriving expressions following this syntax defines a derived role for a given schema. The context sensitive rules, except the rules for "matching", are the same as those for object type deriving expressions. A role deriving expression matches a schema if and only if it obeys the following set of rules. An <object_type_name> in a <shortcut_expression> must be the name of NOLOT or LOT which is the range of the role represented by the <role_name> preceding it. A role represented by a <role_name> in a <shortcut_expression> must be a role of the NOLOT or LOT represented by the <object_type_name> preceeding it, if any. In a conjunction/disjunction of <role_name>s all the role names must be role names of the NOLOT or LOT preceeding the conjunction/disjunction, if any. If there is no LOT or NOLOT, all the role names in a conjunction/disjunction must be of one and the same NOLOT or LOT.

Semantically, a derived role is the serial or parallel composition of the roles it is derived from.

A derived binary relationship type has, as all other binary relationship types, two named roles. A derived binary relationship type can be defined by just one role deriving expression assuming that the other derived role merely follows the **inverse path** of the first one. Except for its name, the inverse of a derived role can always be derived by the interpreter. We analysed, but discarded in example 11, the possibility of defining derived binary relationship types with derived roles that do not follow inverse paths.

Subtype-links can be part of role deriving expressions like any other role. Since subtypes inherit the binary relationship types of their supertypes we only include subtypes in role deriving expressions when we explicitly want to reserve the resulting shortcut for the subtype and prevent it from being used from the supertype.

EXAMPLE 18

Using the meta-schema in figure 6 we can either define the derived role

in_role_called: in ROLE called

and use it both for OBJECT_TYPE, NOLOT, and LOT as follows:

OBJECT_TYPE in_role_called ROLE_NAME='IN_ROLE_CALLED'

LOT in_role_called ROLE_NAME='IN_ROLE_CALLED'

NOLOT in_role_called ROLE_NAME='IN_ROLE_CALLED'

Or, we can reserve this derived role for NOLOTs and LOTs only by defining it as

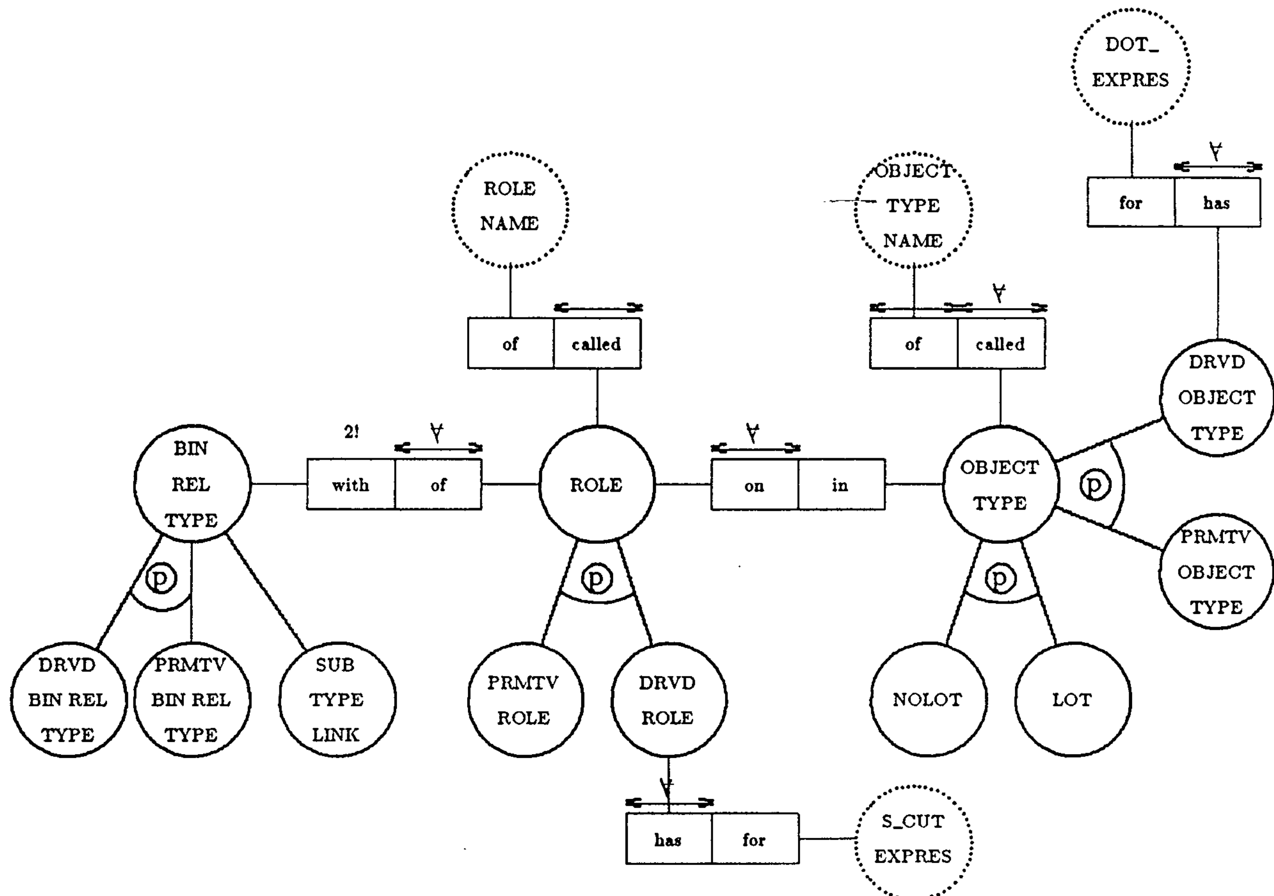
sub_in_role_called: sub OBJECT_TYPE in ROLE called

□

4.5 DOTs and Shortcuts Modelled in the Meta-Schema

Using the graphic formalism for the Binary Relationship Model, we can model the concept of derived object type and shortcut as illustrated in figure 8:

Figure 8. Including the new concepts in the meta-schema



The added part of the meta-schema can be read the following way. ROLEs are partitioned (p) into 'PRIMITIVE_ROLE's and 'DERIVED_ROLE's. Each (∀) 'DERIVED_ROLE' 'has' one (↔) 'SHORTCUT_EXPRESSION', which is 'for' a 'DERIVED_ROLE'. A 'DERIVED_BIN_REL_TYPE' 'is-a' 'BIN_REL_TYPE'. 'OBJECT_TYPE' is partitioned (p) into 'PRIMITIVE_OBJECT_TYPE's and 'DERIVED_OBJECT_TYPE's. Each (∀)

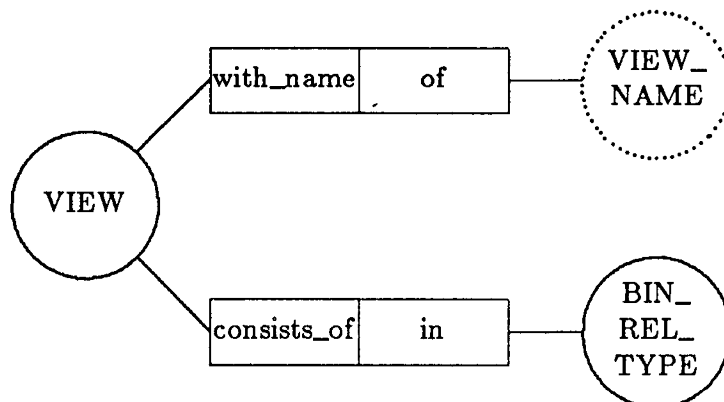
'DERIVED_OBJECT_TYPE' 'has' one 'DOT_EXPRESSION', which is 'for' a 'DERIVED_OBJECT_TYPE'.

Whereas the deriving expression of a derived object type uniquely identifies the object type(s) which are the basis for the derivation, this is not the case for the deriving expression of a derived role. A role, as modeled in the meta-schema, is however unique, so we must require the user to specify the role, and as an implication the object type in that role, whenever he specifies a role deriving expression. This will be discussed in further detail in the next subsection.

Whenever a derived object type or a derived role is used in a query we simply use macro-expansion of the query. This implies that whenever a shortcut expression or a dot expression is made invalid by a change to the parts of the schema - base or derived - from which it is defined, then this will result in an invalid query. We need therefore not impose a lot of constraint checking of shortcut expressions and dot expressions at the time they are defined or during schema change.

We represent the notion of a view in the meta-schema as illustrated in figure 9.

Figure 9. The notion of a view.



EXAMPLE 19

Below we show the transactions which define the view on the meta-schema discussed in example 17..

```
connect new DERIVED_OBJECT_TYPE
    to new NOLOT
    to OBJECT_TYPE_NAME = 'SUB_ROLE'
    to DOT_EXPRESSION = 'ROLE CALLED ROLE_NAME = 'SUB'';

connect new DERIVED_OBJECT_TYPE
    to new NOLOT
    to OBJECT_TYPE_NAME = 'SUP_ROLE'
    to DOT_EXPRESSION = 'ROLE CALLED ROLE_NAME = 'SUP'';

connect new DERIVED_OBJECT_TYPE
```

```
        to new LOT
        to OBJECT_TYPE_NAME = 'DOMAIN_NAME'
        to DOT_EXPRESSION = 'OBJECT_TYPE_NAME';

    connect new DERIVED_OBJECT_TYPE
        to new NOLOT
        to OBJECT_TYPE_NAME = 'IS_A'
        to DOT_EXPRESSION = 'SUBTYPE_LINK';

    connect new DERIVED_BIN_REL_TYPE
        to ( connect new DERIVED_ROLE
                to ROLE_NAME = 'WITH_SUB'
                to SHORTCUT_EXPRESSION =
                    'WITH SUB_ROLE ON OBJECT_TYPE CALLED'
                to DERIVED_OBJECT_TYPE called OBJECT_TYPE_NAME = 'IS_A')
        to ( connect new DERIVED_ROLE
                to ROLE_NAME = 'SUB_OF'
                to SHORTCUT_EXPRESSION =
                    'OF OBJECT_TYPE IN SUB_ROLE OF'
                to DERIVED_OBJECT_TYPE called OBJECT_TYPE_NAME =
                    'DOMAIN_NAME');

    connect new DERIVED_BIN_REL_TYPE
        to ( connect new DERIVED_ROLE
                to ROLE_NAME = 'WITH_SUP'
                to SHORTCUT_EXPRESSION =
                    'WITH SUP_ROLE ON OBJECT_TYPE CALLED'
                to DERIVED_OBJECT_TYPE called OBJECT_TYPE_NAME =
                    'IS_A')
        to ( connect new DERIVED_ROLE
                to ROLE_NAME = 'SUP_OF'
                to SHORTCUT_EXPRESSION =
                    'OF OBJECT_TYPE IN SUP_ROLE OF'
                to DERIVED_OBJECT_TYPE called OBJECT_TYPE_NAME =
                    'DOMAIN_NAME');
```

The first two transactions defined the two derived object types, 'SUB_ROLE' and 'SUP_ROLE', from the object type, 'ROLE', needed for the definition of the derived roles. The next two transactions defined the two derived object types, 'DOMAIN_NAME' and 'IS_A'. The last two transactions defined the two derived binary relationship types, including their derived roles.

We can now finally define the view as follows.

```
    connect new VIEW
        to VIEW_NAME = 'GENERALIZATION_HIERARCHY'
        to BIN_REL_TYPE
          (with ROLE on OBJECT_TYPE
                called OBJECT_TYPE_NAME = 'IS_A'
```

and
 with ROLE on OBJECT_TYPE
 called OBJECT_TYPE_NAME = 'DOMAIN_NAME');

□

As can be seen from the definitions above, a derived object type is only related to the object type it is derived from through its deriving expression. This is acceptable because we replace the name of any derived object type in a DML statement by its deriving expression before we evaluate the statement. If we had modeled the relationship between object type deriving expressions and object types explicitly in the meta-schema, then the system could automatically discard of derived object types made invalid by a change to the object types they are derived from.

Earlier in the paper we made the observation that a role deriving expression may not uniquely identify the role(s) it is composed from, and we promised to explain how to avoid ambiguity. As we see in the example above, the deriving expression for a role is introduced together with the object type in that role in a certain binary relationship type, which solves the problem. Another way to solve this problem, without putting unreasonable restrictions on the use of role names, is to store an ordered set of the roles, the derived role is composed from. This would however require another expansion of the meta-schema and would complicate the definition of derived binary relationship types further.

4.6 The View Update Problem

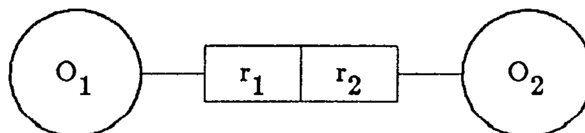
In this subsection we list a few simple rules for unambiguous view update in the Binary Relationship Model extended with the view concepts. The rules are very similar to the rules for view update in the relational model.

Whereas views are extremely useful for data retrieval they usually do not support update. The problem is that the definition of a view alone usually does not provide sufficient information to translate an update on it unambiguously into corresponding updates on the relations the view is derived from. There are various ways of providing extra information at view definition time, one is to let the user choose between a set of possible view update policies as suggested in [Keller 1986], another is to provide the user vith a language that allows him to explicitly define view update operations in terms of operations on the relations the view is derived from as suggested in [Mark 85]. This issue will be discussed in more detail in a forthcomming paper.

When we update through a view, we expect to be able to see the intended effects of the update through the view, and we expect something reasonable to happen to the base of the view.

We shall use the following binary relationship type as the basis for the rules for view update.

Figure 10. Binary relationship type used as basis for view update rules.



RULE 1

If the following update statements are well-defined,

connect O_1	disconnect O_1	reconnect O_1
to O_2	from O_2	from O_2
		to O_3

where O_i, i=1,..,3, are object type names, then the following update statements are well-defined,

connect D_1	disconnect D_1	reconnect D_1
to D_2	from D_2	from D_2
		to D_3

where D_i is the name of an object type derived from O_i, for i=1,..,3.

□

This rule is easily proven by simply substituting the names, D_i, by the corresponding object type deriving expressions. It is important to realize, that the object type deriving expressions can include shortcuts without problems.

RULE 2

Under the same assumption as in rule 1, the following update statements are well-defined.

connect D_1<ref_{D1}>	disconnect D_1<ref_{D1}>	reconnect D_1<ref_{D1}>
to D_2<ref_{D2}>	from D_2<ref_{D2}>	from D_2<ref_{D2}>
		to D_3<ref_{D3}>

If the deriving expression for the derived object type, D_i, is "O_i <ref_{Oi}>" then we substitute "D_i <ref_{Di}>" by "O_i (<ref_{Oi}> and <ref_{Di}>)".
A similar generalization can be made when <ref_{Di}> or <ref_{Oi}> is a conjunction/disjunction of <ref>s.

□

RULE 3

Under the same assumptions as in rule 1, let the derived binary relationship type

$$O_1\ (d_1,d_2)\ O_2$$

have the role deriving expressions

$$d_1: r_1$$
$$d_2: r_2$$

Then the following update statements are well-defined for this derived binary relationship type.

connect S_1	disconnect S_1
to S_2	from S_2

A similar rule applies to the reconnect statements.

□

RULE 4

There are no other well-defined update statements on derived binary relationship types.

□

For an informal proof, consider the following base binary relationship types

$$O_1 \; (r_1, r_2) \; O_2 \; (s_2, s_3) \; O_3$$

and the following derived binary relationship type

$$O_1 \; (d_1, d_3) \; O_3$$

with the role deriving expressions

$$d_1: r_1 \; O_2 \; s_2$$
$$d_3: s_3 \; O_2 \; r_2$$

And, consider the following update statements on this derived binary relationship type

connect O_1	disconnect O_1
to O_3	from O_3

The base binary relationship types can be connected in a variety of ways to give the desired effect of the update when seen through the derived binary relationship type. We could connect O_1 to O_3 via new elements in O_2, or via elements in O_2 already connected to O_1, or via elements in O_2 already connected to O_3, or via elements in O_2 not yet connected to O_1 or O_3. None of these can in general be guaranteed to be reasonable solutions.

The base binary relationship types can be disconnected in a variety of ways to give the desired effect of the update when seen through the derived binary relationship type. We could disconnect all elements in O_2, currently connecting O_1 to O_3, from O_1 and from O_3; or we could disconnect them only from O_1; or we could disconnect them only from O_3; or we could disconnect some of them from O_1 and the rest from O_3. None of these can in general be guaranteed to be a reasonable solution.

If we want to be able to update derived binary relationship types in general, then we have to provide more information when we define them. In this respect, our view concept is no different from others. One of the alternatives is to choose one of the solutions above to be applied in all cases. Another is to specify for each derivation which of the above solutions applies. Another is to make the system ask the user for sufficient information to make an update well-defined.

5. Summary

After a brief presentation of the Binary Relationship Model, we informally introduced two new concepts - derived object types and shortcuts - for view definition in this model. We continued with a formal definition of the concepts in the context of a self-describing meta-schema for the Binary Relationship Model and we extended the meta-schema to integrate the new concepts in the model. We demonstrated how the DML can be used to define and change views through the self-describing meta-schema. We finally provided a set of simple rules for updates through views defined in terms of the new concepts.

Our work represents to our best knowledge the first comprehensive theory of views for the Binary Relationship Model.

Much interesting and important work remains, especially on the view update problem. Another interesting research topic is to generalize our view concepts to support the definition of

recursive views.

One of the problems with the Binary Relationship Model is that schemata defined in the model give a "flat" representation of information, making it hard to identify aggregates. If we could just "color" a schema emphasizing the aggregates defined in it ....

One of the nice things about the Binary Relationship Model is that it supports subtyping. We could however improve the model by generalizing it to support subtyping per category, e.g. female and male are the subtypes of person wrt. the category sex, and secretary, engineer, programmer, etc. are subtypes of person wrt. the category job.

References

[Abrial 74]:

Abrial J.R., "Data Semantics", in Klimbie & Koffeman (Eds.), Data Base Management, North-Holland, 1974, pp.1-59.

[Breutman, Falkenberg, and Mauer 79]:

Breutman, Falkenberg E., and Mauer, "CSL: A Conceptual Schema Language" Proceedings IFIP TC 2 - WG 2.6 Working Conference, Munchen, Germany, March 1979.

[Buneman and Frankel 79]:

Buneman P. and Frankel R., "FQL - A Functional Query Language", ACM SIGMOD International Conference on Management of Data, 1979.

[Ceri, Pelagatti, and Bracchi 81]:

Ceri S., Pelagatti G., and Bracchi G., "Structured Methodology for Designing Static and Dynamic Aspects of Data Base Applications, Information Systems, Vol. 6, No. 1, 1981.

[Hall, Owlett, and Todd 76]:

Hall P., Owlett J., Todd, S., "Relations and Entities" in Nijssen (Ed.) Modelling in Database Management Systems, North-Holland, 1976.

[ISO 82]:

Griethuysen (Ed), "ISO TC97/SC5/WG3: Concepts and Terminology for the Conceptual Schema and Information Base", American National Standards Institute, 1982.

[Keller 86]:

Keller, A.M., "The Role of Semantics in Translating View Updates," IEEE Computer, January 1986.

[Kent 78]:

Kent W., Data and Reality, North-Holland, 1978.

[Mark 83]:

Mark L., "What is the Binary Relationship Approach?", In Davis (Ed.) Entity-Relationship Approach to Software Engineering, North-Holland, 1983.

[Mark and Roussopoulos 83]:

Mark L. and Roussopoulos N., "Integration of Data, Schema, and Meta-Schema in the Context of Self-Documenting Data Models", in Davis (Ed.), Entity Relationship