ESPRIT Project 6353 (NATURE), Report SI-92-01

Concept Formation by Reverse Modelling
- Preliminary Study

Petia Assenova
SISU The Swedish Institute for Systems Development
Box 1250, S-164 28 Kista, Sweden
petia@sisus.sisu.se

ABSTRACT. In this report, we study how relational schemas can be translated into conceptual schemas. We first give an overview of the modelling language ConceptBase. To illustrate how ConceptBase can be used we include two models. The first one is a meta model of the relational model, and the second is a meta model of a conceptual modelling language. In the meta models, we have focused on those aspects that are relevant for the translation of a relational schema into a conceptual one.

0 INTRODUCTION

Translating traditional data models to conceptual models is an important topic in the area of reverse engineering. In the context of information systems engineering, the process of reverse engineering starts from an existing system and analyses it in order to identify its components and their relationships. Further, a description of the system at a higher level of abstraction is created. This description is often expressed in an object-based conceptual modelling language, where it is possible to handle declarative enterprise rules. Reverse engineering usually includes a knowledge acquisition process, which adds semantically information not contained in the original system. The interest in reverse engineering has increased recently due to the large number of legacy systems presently in operation. Often, the exact understanding of the data and procedures in the existing systems has been lost after years of evolution, and reverse engineering is seen as a means to regain that understanding.

In the NATURE project, we envisage that a tool for reverse modelling can be a component of the Requirements Capture Assistant.

1 ABOUT CONCEPTBASE

In this section we will describe the basic concepts of ConceptBase. We will present the examples in three forms- 1) graphically; 2) by means of the user environment (show how the user interacts with ConceptBase); and, 3) by the internal representation (show how the information is represented in ConceptBase).

ConceptBase is a language combining logic and object oriented constructions. The internal representation of the information is like propositions in logic, and at the same time the thinking is object oriented. The information is structured as objects, called *individuals* .

Ex 1:1.

Fig 1:1 Graphical presentation of individuals in ConceptBase

User environment

Individual Eva with end Eva (where Eva is just a girl name)
Individual SimpleClass with end SimpleClass (where SimpleClass is an abstract concept)

Internal representation

proposition(Eva,Eva,-,Eva,...)
proposition(SimpleClass,SimpleClass,-,SimpleClass,...)

A proposition has the form: 'proposition(id,x,l,y,t,tt)', where id is an identifier, x is called a source, l is called a label, y is called a destination, t is called a validity time and tt is called a belief time. We will not cover t and tt in this paper.

An individual can have connections with other individuals. In the terminology of ConceptBase a connection is called an attribute.

Ex 1:2

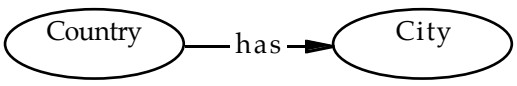


Fig 1:2 Graphical presentation of attributes in ConceptBase

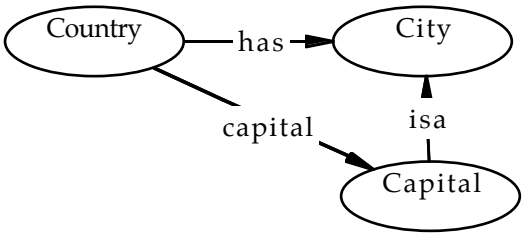
The information in fig 1:2 represented in ConceptBase:

Individual Country with
attribute
has : City
end Country
Individual City with end City

Internal representation of the declarations above:
proposition(Country,□Country,□-,□Country,...)
proposition(#1,□Country,□has,□City,...)
proposition(City,□City,□-,□City,...)

An object can have specialisations. This means that if an individual A is a specialisation of the individual B, then A inherits all properties which B possesses.

Ex 1:3

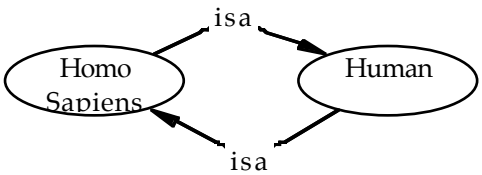


We can add the following to the declarations in ex 1:2:
Individual Country with
attribute
capital : Capital
end Country
Individual Capital isA City with
end Capital

Fig 1:3 Graphical presentation of specialisations (isA) in ConceptBase

Internal representation:
proposition(#2,□Country,□capital, Capital,...)
proposition(Capital,Capital,□-□Capital,...)
*proposition(#3,□Capital,□*isa*[1] ,□City,...)*

Note that we can add attributes to already existing individuals. Consequently an individual can be declared in several separate declarations.



Furthermore the situation that A is a specialisation of B and B is a specialisation of A at the same time, where A and B are individuals, is possible.(Fig 1:4)

Fig 1:4 Example of specialisations

Ex 1:4
User interface
Individual A with end
Individual B isA A with end
Individual A isA B with end

We can not write:
Individual A isA B with end
Individual B isA A with end
This is because we can not declare A as a specialisation of an undeclared object, and B is not declared at this time. Consequently the order of the declarations is significant.

An individual can have instances. If the individual A is an instance of the individual B, then A has the same properties as B. Furthermore, the Instantiation axiom says that if p is a proposition which is an instance of proposition P, then the source of p must be an instance of the source of P, and the destination of p must be an instance of the destination of P. [Jarke91]

[1]The asterisk in the beginning of isa is a syntax detail

Ex 1:5

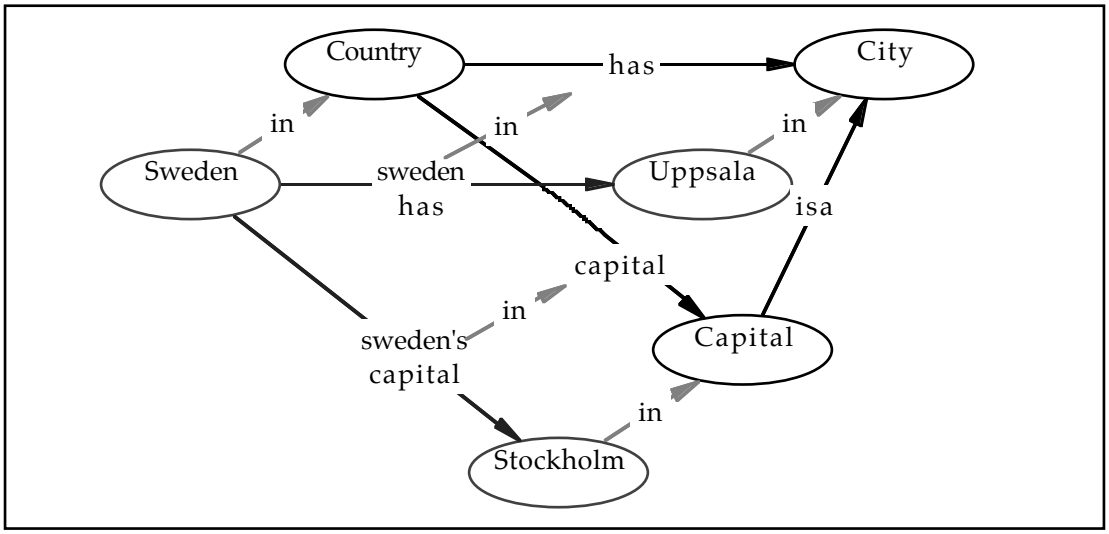


Fig 1:5 Graphical presentation of instances (in) in ConceptBase

User environment:

Individual Sweden in Country with
has
    sweden_has : Uppsala
capital
    Sweden's_capital : Stockholm
end Sweden
Individual Stockholm in Capital with end Capital
Individual Uppsala in City with end Uppsala

Internal presentation

proposition(Sweden,Sweden,□-□,Sweden,...)
proposition(Uppsala,Uppsala,□-□,Uppsala,...)
proposition(Stockholm,Stockholm,□-□,Stockholm,...)
*proposition(#4,Sweden,*instanceof,Country,...)*
*proposition(#5,Uppsala,*instanceof,City,...)*
*proposition(#6,Stockholm,*instanceof,Capital,...)*
proposition(#7,Sweden,Sweden_has,Uppsala,...)
proposition(#8,Sweden,Sweden's_capital,Stockholm,)
*proposition(#9,#7,*instanceof,#1,...)*
*proposition(#10,#8,*instanceof,#2,...)*

'Sweden_has' is an attribute and an instance of the attribute 'has'. If we do not write something in the place of 'sweden_has', ConceptBase generates a unique value, which identifiers the missing attribute.

One interesting fact is that if A and B are two separate individuals; the situation that A is an instance of B and B is simultaneously an instance of A, is still possible. Consequently, A can be an instance of A.

Ex 1:6 The declaration of the predefined individual Proposition
Individual Proposition in Proposition isA Proposition with
attribute
    ...
    attribute : Proposition
    ...
end Proposition

ConceptBase provide certain predefined individuals. Some of them are *Token, SimpleClass, MetaClass and MetametaClass*, and we can use them to structure the information (i.e. individuals defined by us) hierarchically. The declaration of them is analogous to the declaration of SimpleClass in ex 1:1. There is not a connection between those, which we can see, but there exists an order between them. If Token is at level n, so SimpleClass is at level n+1 and so on. This confirms the fact that an individual which is an instance of Token has no instances itself, and that an individual which is an instance of SimpleClass has instances of Token as instances, an individual which is an instance of MetaClass has instances of SimpleClass as instances [Mylopoulos90].

There are some constraints on the usage of hierarchies. Let A and B be two individuals which are instances of the classes enumerated above. B can be an instance of A only if B is one level lower than A. B can be a specialisation of A only if B is at the same level as A. If just A is an instance of one of the classes enumerated above and B is either an instance of or a specialisation of A, then B is placed automatically at the same level as A or one level lower than A respectively.
Ex: 1:7

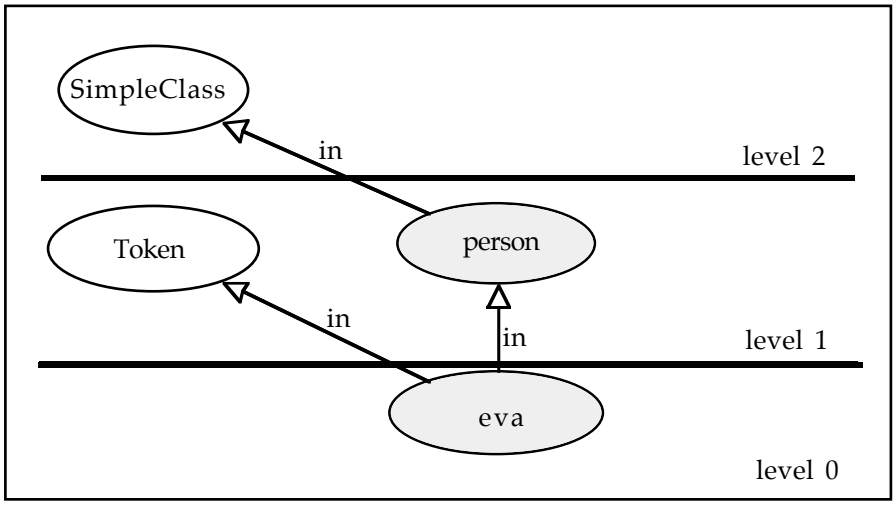


Fig 1:6 Graphical presentation of example 1:7

The connection 'person in SimpleClass' puts 'person' one level lower than SimpleClass. Then 'person' is at the same level as Token and 'Eva' can be an instance of both.

User environment
Individual Person in SimpleClass with end
Individual Eva in Token,Person with end

Internal representation
*proposition(#11,Person,*instanceof,SimpleClass,...)*
*proposition(#12,Eva,*instanceof,Token,...)*
*proposition(#13,Eva,*instanceof,Person,...)*

However the hierarchical structure can be omitted if we work with a model with only one level.

There is another dimension in which we can structure information. The concepts *Class, QueryClass, GenericQueryClass* and *Magic* are used to show the field of application of the individuals. We will not describe the concepts in detail or discuss their syntax, but only give an overview and some examples.

Individuals which are instances of Class may define deductive rules or integrity constraints [Jarke91].
Ex 1:8 A constraint that says that a country can have just one capital.

```
Class Country with
        bdmconstraint
                Country_constraint : $ forall c/Country
                                        forall d,e/Capital
```

```
                                        ((AttrValue(c,capital,d,Always) and
                                         AttrValue(c,capital,e,Always))
                                        ==> EQUAL(d,e)
                                        ) $
end Country
```

Note that it is not necessary to write Individual in the beginning of the declaration. It is generated automatically.

In QueryClass, we define the queries regarding which we want to ask the knowledge base.
Ex 1:9 A question about all countries and their capitals.

```
QueryClass Countries_and_their_capitals isA Country with
retrieved_attribute
        capital : Capital
end Countries_and_their_capitals
```

Ex 1:10 A question about the capital of Sweden.

```
QueryClass The_Capital_of_Sweden isA Capital with
query_rule
        The_Capital_of_Sweden_rule : $
                AttrValue(Sweden,capital,this,Always) $
end The_Capital_of_Sweden
```

GenericQueryClass is a kind of QueryClass and we look at this as a procedure in an imperative language. We can send variables and constants when we call them.
Ex 1:11

```
GenericQueryClass Country_and_its_capital isA Capital with
        parameter
                c : Country
        query_rule
                Country_and_its_capital_rule : $
                        AttrValue(c,capital,this,Always) $
end Country_and_its_capital
QueryClass Derived_Country_and_its_capital isA
                Country_and_its_capital([Sweden/u])
with end
```

When working with recursion, we use the class Magic.
Ex 1:12

```
Magic,QueryClass X isA Y with
query_rule
        X_rule : $ AttrValue(a,b,this,Always)
                        and
                exists y/Y
                InstanceOf(y,X,Always) and
                AttrValue(y,b,this,Always) $
end X
```

An Individual can simultaneously be an instance of Class, Magic and some of the hierarchy classes.

2 A CONCEPTUAL SCHEMA OF A RELATIONAL DATABASE SCHEMA IN CONCEPTBASE

To describe the translation from a relational to a conceptual schema, we will construct meta models of the relational model and a conceptual modelling language. We will also show how objects are translated from one model to another. In this section we will present the basic concepts of the relational model and construct a conceptual model of these. In section 3 we will make the same for a conceptual modelling language.

2.1 Basic Concepts

The relation scheme $R_i(X_i)$ is a pair $<R_i,X_i>$, where R_i is the relation scheme name and X_i denotes a sequence of columns: $X_i=C_{i1},C_{i2},...,C_{im}$, $m \in N$ (the set of natural numbers). A column, often denoted C_{ij}, is a pair $<C_{ij},D_{ij}>$, where C_{ij} is the name of the column and D_{ij} is a set, called a domain of the column. A tuple for the relation scheme $R_i(X_i)$ is an element of $D_{i1}\times\ldots\times D_{im}$. A relation for a relation scheme $R_i(X_i)$ is a set of tuples for $R_i(X_i)$. If t is a tuple for $R_i(X_i)$ and Y is a subsequence (a part of a sequence) of X_i, then t[Y] denotes the subtuple of t corresponding to Y.

Let $R_i(X_i)$ be a relation scheme, r_i a relation for $R_i(X_i)$ and Y and Z be subsequences of X_i. The projection of r_i on Y, denoted by $\pi Y(r_i)$ is $\{t[Y] \mid t \in r_i\}$. A functional dependency over $R_i(X_i)$ denoted $Y \rightarrow Z$ is satisfied where for any two tuples t and t' in r_i, $t[Y]=t'[Y]$, implies $t[Z]=t'[Z]$. A key for $R_i(X_i)$ is a subsequence of X_i, K_i, such that $K_i \rightarrow X_i$ is satisfied for any t_i associated with $R_i(X_i)$ and there does not exist any proper subsequence of K_i having this property. A relational scheme can have several candidate keys from which one primary key, denoted by PK_i, is chosen.

Let $R_i(X_i)$ and $R_j(X_j)$ be two relation schemes and r_i and r_j be relations of $R_i(X_i)$ and $R_j(X_j)$, respectively. An inclusion dependency, denoted $<ID,[R_i.Y_i]<<[R_j.Y_j]>$, where ID is the name of the inclusion dependency, Y_i and Y_j are subsequences of X_i and X_j respectively, is satisfied by r_i and r_j if $\pi Y_i(r_i) \subset \pi Y_j(r_j)$.

A relational schema is the quadruple <RS,R,F,I>, where RS is the name of the relational schema, R is a set of relation schemes, F is a set of functional dependencies and I is a set of inclusion dependencies. A database state of RS is a set of n relations, $\{r_1,...,r_n\}$, where r_i is a relation for R_i. An example of a Relational Database follows.

Ex 2:1 Student - examination schema and database state

Student

Person nr	Name	Address	Telephone nr
111	Eva	Stockholm	123 456
222	Kalle	Stockholm	135 246
333	Ann	Uppsala	987 654

Course

Code	Name	Points
ASY1:A	Pascal	7
ASY2:E	Simula	6

Examination

Student*	Course*	Date	Mark
111	ASY1:A	19901010	5
222	ASY1:A	19901010	4
111	ASY2:E	19911108	4

The underlined words are primary keys.
The words with asterisk are foreign keys.

In the example above we have a relational schema RS=Student-examination schema. The set R of relation schemes is {<Student,[Person nr,Name,Address,Telephone nr]>, <Course,[Code,Name,Points]>, <Examination,[Student,Course,Date,Mark]>}. The columns Person nr, Points, Date and Mark have Integer as their domain and the other columns have String as their domain. An example of a tuple in student is <111,Eva,Stockholm,123 456>. The set {<ASY1:A,Pascal,7>,<ASY2:E,Simula,6>} is a relation for the relation scheme Course. An example of a database state of the relational schema Student-Examination contains all the relations in the figure. A set F of functional dependencies is {<Person nr → Name, Address,Telephone>, ...,

<Person nr,Name,Address→Telephone>, ...,<Student,Curse→Date,Mark>,...
A set I of inclusion dependencies is {<id1,Examination.Student<<Student.Person nr>, <id2,Examination.Course<<Course.Code>}.

2.2 Graphical Presentation

Figure 2:1 presents a conceptual model of the relational model in ConceptBase. We have used some "help" objects which are not described above. A cell is the smallest unit in our model of the relational model. The reason to introduce it is that the value of a cell is indivisible. Each cell has a value. Example: In Student - examination schema, Simula is the value of the cell in the column Name (in relation scheme Course) in the tuple <ASY2:E,Simula,6>. Note that with the concept cell we do not need to take care of the order in the column sequence because there is already a connection between a cell and a tuple and between a cell and a column.

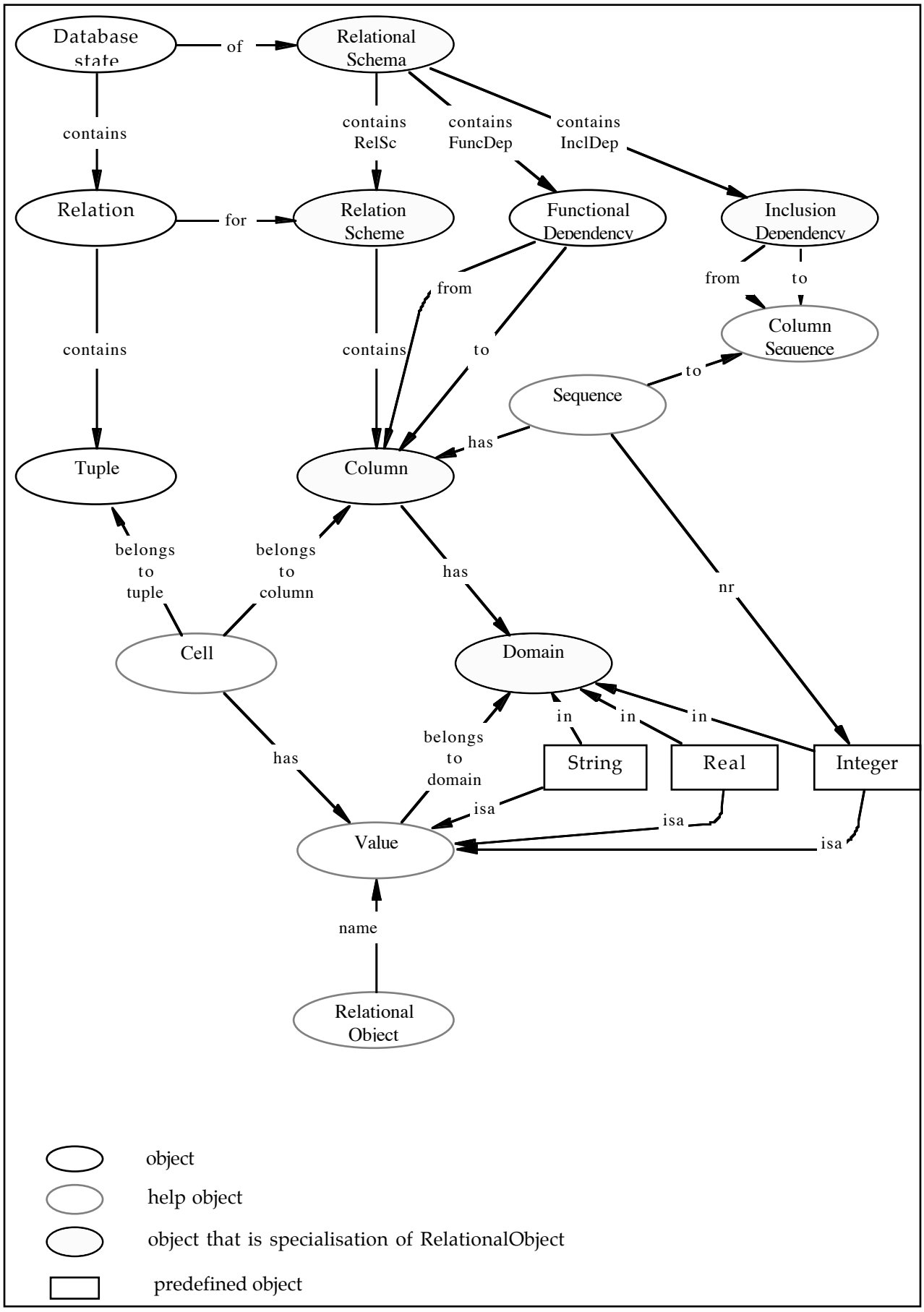


Fig 2:1 A conceptual schema of the relational database model

The objects sequence and column sequence are included to enable the use of lists, which are necessary for the representation of inclusion dependencies, because the order of columns in an inclusion dependency is important.

The concept RelationalObject is a generalisation of the objects' RelationalSchema, RelationScheme, Column, Domain and InclusionDependency. The common property is that each of these objects have a name. Sometimes it can be relevant to have other properties such as date of creation, authority and so on, which can be connected to RelationalObject and inherited by the specialisations (isA links) of it.

We view the predefined objects String, Real and Integer as instances of the object Domain. Since the destination is one level higher than the source, the object Domain must be an instance of MetaClass. It is possible to define other instances of Domain such as all positive integers if necessary. Furthermore, we look at those objects as specialisations of the object Value. Also the number seven, which is an integer, is an instance of Value. The fact that the destination and the source must be in the same level entails that the object Value must be an instance of SimpleClass.

The usage of the object Relation can be discussed. Will Relation be included in the model or not? Consider the statements : 1)□Let r_i be a relation; and, 2)□Let r_i be a set of tuples. Statement 1 says more than statement 2, namely that all tuples in r_i belong to one relation scheme. A Tuple belongs to a Relation for a RelationScheme which is the same as to say that a Tuple belongs to a RelationScheme. We can create the attribute 'belongs to RelSc' between the last two objects. Then the relation r_i can be computed from the information about the tuples which already exists. Consequently the object Relation contains no more information and it is therefor not necessary and can be omitted (fig 2:2).

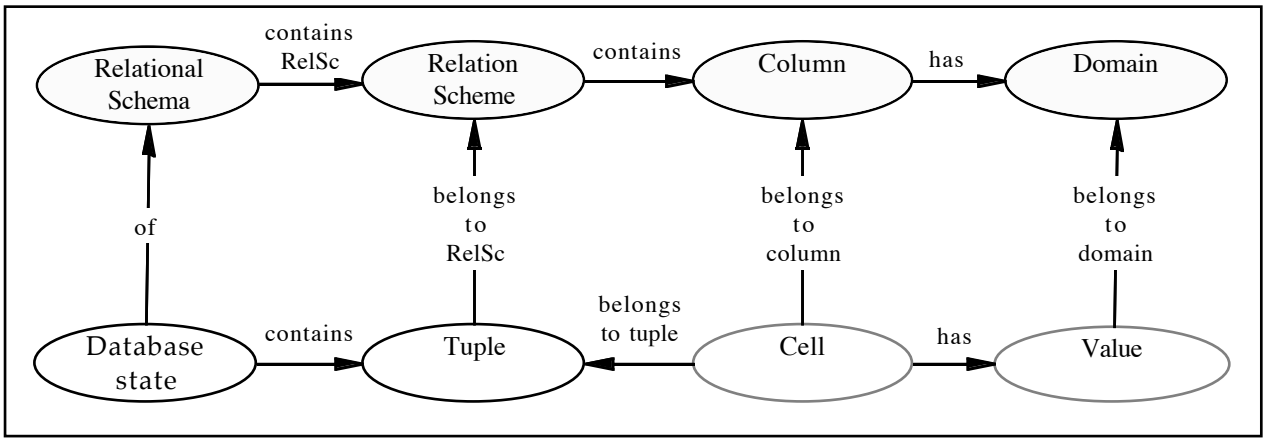


Fig 2:2 A part of conceptual schema of relational model without the concept Relation

Another way to view Database state, Tuple, Cell, and Value is as instances of Relational Schema, Relational Scheme, Column and Domain respectively. E.g. <tuple1,[111,eva,stockholm,123 456]> is an instance of the relation scheme <student,[person nr, name,address,telephone]>. The disadvantage of this point of view is that both Integer and Eva for example become instances of Domain. Then it is natural to apprehend Eva and Integer as the same kind of objects. But the meaning of Eva has nothing in common with the meaning of Integer.

3 A CONCEPTUAL SCHEMA OF A CONCEPTUAL MODEL

In this chapter we will, in ConceptBase, model a conceptual schema. We use Moloc (**Mo**delling in **Lo**gi**c**) as an example of a first order language for conceptual modelling. A complete description of Moloc can be found in [Boman92]. The definitions below are standard in conceptual modelling.

3.1 Basic Concepts

Let P and S be two sets of symbols. A (first order) language based on <P,S>, written L(P,S) is defined on an alphabet consisting of connectives, quantifiers, punctuation symbols, variables, constants S, and predicate symbols P, where each predicate symbol has an arity. A term is a constant or a variable. An atom is a formula of the form $p(t_1,...,t_n)$ where p is a predicate symbol and $t_1,...,t_n$ are terms.

An integrity constraint is defined as any closed first order formula. Some special cases of constraints are typing-, mapping- and generalisation constraints. A typing constraint is a formula of the form $\forall x \forall y(A(x,y) \Box\rightarrow D(x))$ or the form $\forall x \forall y(A(x,y) \Box\rightarrow R(y))$. A formula of the first form will be read as 'the domain of A is D' and a formula of the second form as 'the range of A is R'. Mapping constraints concern the cardinality of attributes and have one of the following four forms. The expression 'the attribute A is single-valued' is an abbreviation of the formula $\forall x \forall y \forall z(A(x,y) \wedge A(x,z) \Box\rightarrow y \Box= \Box z)$. The expression 'the attribute A is injective' is an abbreviation of the formula $\forall x \forall y \forall z(A(y,x) \Box\wedge A(z,x) \Box\rightarrow y \Box= \Box z)$. The expression 'the attribute A is total' is an abbreviation of the formula $\forall x(D(x) \Box\rightarrow \exists y A(x,y))$. The expression 'the attribute A is surjective' is an abbreviation of the formula $\forall x(R(x) \Box\rightarrow \exists y A(y,x))$. A generalisation constraint is a formula of the form $\forall x(P(x) \Box\rightarrow Q(x))$ and reads $P \Box\subset \Box Q$

A conceptual schema is a pair <L,C>, where L is a language and C is a set of constraints. It is assumed that all predicate symbols in L are unary or binary, and that for each binary predicate symbol p, there exist two typing constraints in C specifying the domain and range of p. We assume there exist a set of predicate symbols LP and a set of constant symbols LC (called lexical predicate and constant symbols). An example of LP is {String, Integer, Real} and an example of LC is {abc, 7}.

The definitions above are from [Johannesson92].

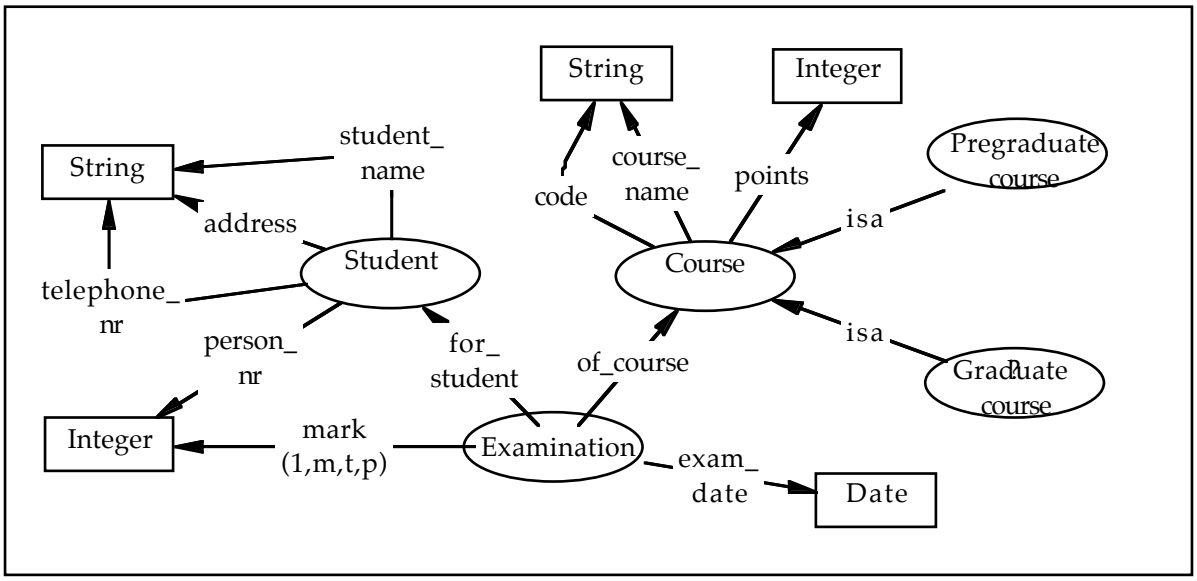


Fig 3:1 An example of a conceptual schema.

An example of a conceptual schema is shown graphically in fig 3:1. The binary predicates are {person_nr, person_name, address, telephone_nr, code, course_name, points, for_student, of_course, mark, exam_date}. The unary predicates are Date, Integer, String, Student, Course, Pregraduate_course, Graduate_course, Examination, where the first three are lexical types and the others are entity types. The generalisation constraints are shown by 'isa' and in the example Course is generalisation of Pregraduate_course and Graduate_course. (The predicates Pregraduate_course and Graduate_course are not necessary in this small example. We include them because specialisation is an important concept in conceptual modelling which does not exist in the relational model.) The typing constraints are shown with directed arrows. The beginning of the arrow shows the domain of the predicate and the end of the arrow shows the range of the predicate. As an example, the domain of mark is Examination and the range of mark is Integer. The binary predicate Mark has the mapping constraints (1,m,t,p). The symbols in the first and second positions in the parentheses show that the predicate is single-valued denoted by 1 (m for not single-valued) and not injective denoted by m (1 for injective) respectively. The symbols in the third and fourth positions show that the predicate is total denoted by t (p otherwise) and not surjective denoted by p (t for subjective) respectively. Note that the graph contains just a part of the concepts of the conceptual schema. In Moloc we do not model the instance level graphically. All instances are represented in the information base. An example of information base to the conceptual schema in fig 3:1 is following below.

```
student(s1).
student(s2).
student(s3).
student_name(s1,'Eva').
person_nr(s1,111).
address(s1,'Stocholm').
...
pregraduate_course(pc1).
code(pc1,'ASY1:A').
course_name(pc1,'Pascal').
points(pc1,7).
...
examination(e1).
for_student(e1,s1).
of_course(e1,pc1).
mark(e1,5).
exam_date(e1,19901010).    ...
```

We call the constraint which is not typing or mapping or generalisation constraint for User Defined constraint. Example: a student can not examine a course a second time if he has mark greater or equal then three. In Moloc code this will be written: inconsistent :- for_student(E1,S), for_student(E2,S), of_course(E1,C), of_course(E2,C), mark(E1,M), not(M<3), not(E1=E2). Such general kinds of constraint do not exist in the Relational model.

3.2 Graphical Presentation

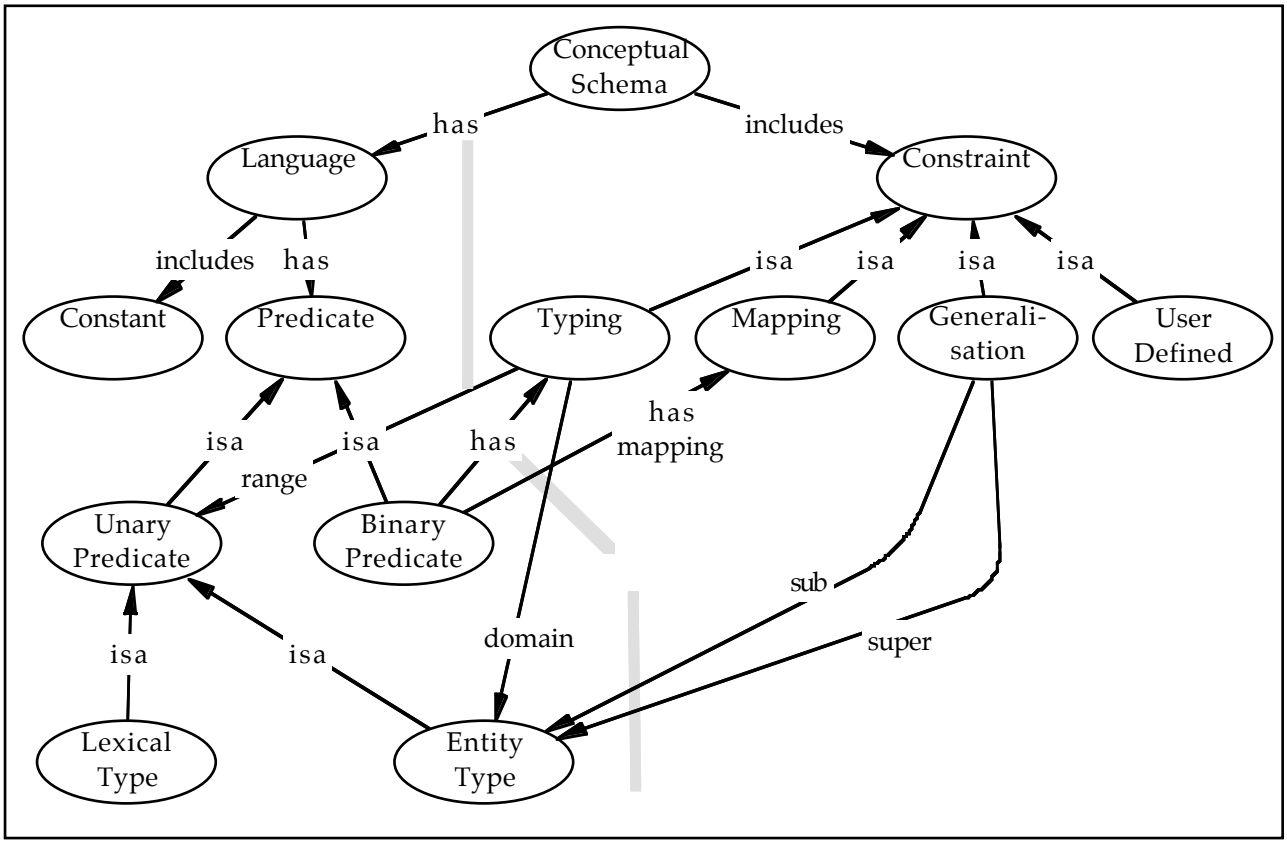


Fig 3:2 A conceptual schema of conceptual schema

All concepts in fig 3:2 are at the same level. This is because the instance level (the level which includes all instances) is not represented in the model. The schema above can be seen as the integration of a part with statements (on the right) and a part with symbols (on the left). Another interesting fact is the level of the specialisations. (Entity Type and Lexical Type are specialisations of Unary Predicate witch is a specialisation of Predicate.) A consequence of this is a long path if we traverse the graph from Entity Type to Conceptual Schema.

As illustrated in fig 3:1 the graphical presentation of a conceptual schema does not include all concepts described above. Below, we present a meta model that only includes those parts of a conceptual schema that can be expressed in a graphical representation. The object Constant is relevant only when we are working with the instance level. It is not necessary when modelling the type level (the level which describes the object types and the relation between them) and can therefore be omitted. The objects Typing and Generalisation can be reduced if we connect the object Binary predicate with Unary Predicate and Entity Type and call the connecting attributes range and domain respectively. We will not get any instances of User Defined after translations of a relational schema to a conceptual schema. Further, it is not necessary to include the object User Defined. Then we do not need the object Constraint because we have just one specialisation of it. Finally, we omit the objects Language and Predicate for the sake of simplicity. Se fig 3:3. The advantage of the reduced schema is that it

include exact everything that we have in a graphical conceptual model. The disadvantage is that we lose the classification of statements and symbols.

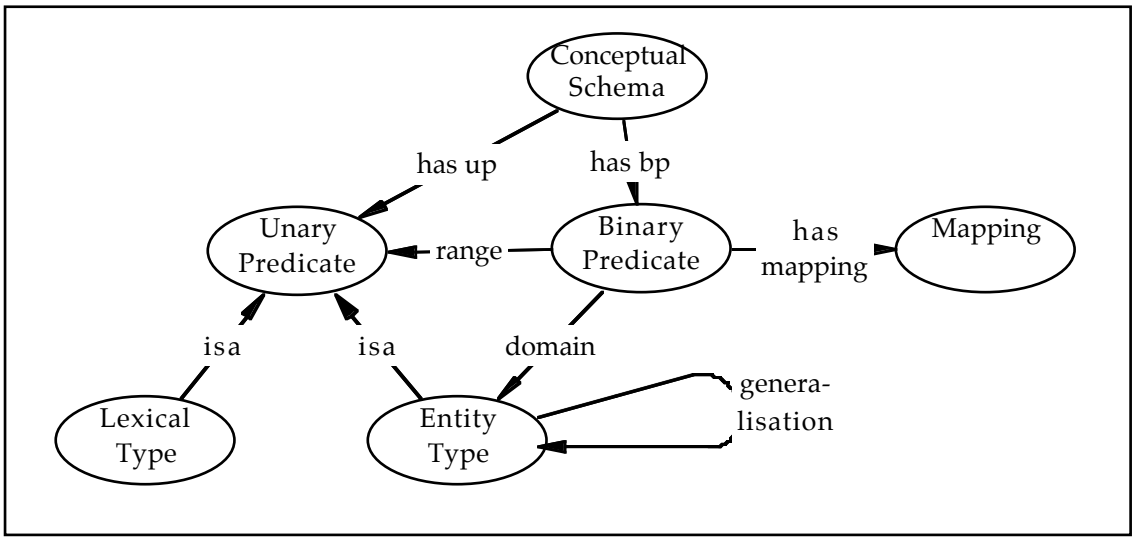


Fig 3:3 A reduced schema to the schema in fig 3:2

We have remarked on some differences between the relational model and conceptual models. We will discus another important distinction between the two models. Consider haw the information for Eva as example is stored in both schemas. In a relational database this will be <111, Eva, Stockholm, 123 456> which is matching with <Person nr, Name, Address, Telephone nr>. The same information in an information base for a conceptual schema is represented as student(s1), student_name(s1,'Eva'), person_nr(s1,111) and so on. In the second case we have an object identifier s1 for a student's object with properties name Eva and person nr 111. Name and person nr are properties of a student. Eva and 111 are values describing the properties. We can change values without to change the identifier s1 of the object. In the first case we have just values. Usually some of them are using as identifier. Then a value can both be an identifier and a description of a property. It is also possible that the changing of the description of a property results in a change identifier.

4 ALGORITHM FOR TRANSLATION

The reverse modelling method we are currently implementing is described in [Johannesson92]. In this section, we give an informal overview of the method proposed. It is shown in the paper that the translation process cannot be completely automated, since a conceptual schema can carry more semantic information than a relational one. A systematic method for acquiring the extra domain knowledge needed in the translation process is given.

A basic assumption of the method suggested is that all relations given shall be in the third normal form, and that all keys and inclusion dependencies shall be specified. The first steps of the method transform the relational schema into a form appropriate for identifying object structures Roughly speaking, the transformations aim at obtaining a one-to-one relationship between relations and object types by splitting and collapsing relations. Normally, a relation corresponds to a single object type. However, if a relation contains several keys, it may correspond to more than one object type. For example, the relation

COUNTRY(Name, Capital, Population, Capital_population) corresponds to two object types: COUNTRY and CAPITAL (we assume that Capital is a candidate key in COUNTRY). If a user decides that a relation corresponds to several object types, then it is split into several relations. Another case where a single relation may correspond to several object types is when there exists an inclusion dependency whose right hand side is not a key. In such a case, the right hand side of the inclusion dependency may correspond to an additional object type, as in the following example:

CITY(Name, Country, Population)
PERSON(SS#, Lives_in)
PERSON.Lives_in << CITY.Country

The inclusion dependency above indicates that there should exist an object type COUNTRY, which does not correspond to any of the original relations. In such a situation, a new relation is obtained by splitting the relation occurring on the right hand side of the inclusion dependency.

In some cases, several relations may correspond to a single object type. This occurs when the relations are involved in a cycle of inclusion dependencies. In order to obtain a one-to-one relationship between relations and object types, the relations will in such a case be collapsed into a single one. When the transformations outlined above have been applied, the resulting relational schema is translated into a conceptual one by mapping each relation into an object type and each inclusion dependency into either a generalisation relationship or an attribute. It should be noted that the method does not in all cases automatically determines if an inclusion dependency corresponds to a generalisation relationship or an attribute; this must sometimes be determined by a user with knowledge of the semantics of the relational schema.

REFERENCES

[Boman92] Boman M, Bubenko J, Johannesson P "Conceptual Modelling" third ed, Department of Computer and Systems Sciences, Stockholm University, Sweden, 1992

[Jarke91] Matthias Jarke "ConceptBase V3.0 User Manual" MIP-9106, Fakultät für Mathematik und Informatik, Universität Passau, Germany, 1991

[Johannesson92] Paul Johannesson "A Method for Transforming Relational Schemas into Conceptual Schemas", Department of Computer and Systems Sciences, Stockholm University, Sweden, 1992

[Mylopoulos90] Mylopoulos J, Borgida A, Jarke M, and Koubarakis M "Telos: A Language for Representing Knowledge about Information Systems", MIP-9011, Fakultät für Mathematik und Informatik, Universität Passau, Germany, 1990